INTELLIGENT ELECTRONIC DESIGN FOR MECHATRONIC SYSTEMS

SAMIR A. EL-NAKLA

A thesis submitted in partial fulfilment of the requirements
of the University of Abertay Dundee for the degree of
Doctor of Philosophy

September 2004

INTELLIGENT ELECTRONIC DESIGN FOR MECHATRONIC SYSTEMS

SAMIR A. EL-NAKLA

A thesis submitted in partial fulfilment of the requirements
of the University of Abertay Dundee for the
degree of Doctor of Philosophy

September 2004

I certify that this thesis is the true and accurate version of the thesis approved by the examiners.

Signed. [redacted] Date.....15/11/04.....

(*Director of studies*)

Declaration

I hereby declare that while registered as a candidate for the degree for which this thesis is presented I have not been a candidate for any other award. I further declare that, except where stated, the work in this thesis is original and was performed by myself.

Signed [redacted] (Samir A. El-Nakla) Date. 12.11.04.....

Acknowledgements

I would like to express my sincere gratitude to my supervisor Professor David Bradley for his guidance, support, continued encouragement, enthusiasm and kindness. Also I would like to thank my second supervisor Dr Allan Macleod for his assistance when I needed him and for revising my thesis.

My thanks also to Mr Allan Milne and Victor Bassilious for helping me writing the software. Thanks also to every member of staff at the University of Abertay who kindly helped me during the project.

Dedication

This work is dedicated to my wife Darin and my kids for their support and patience throughout the duration of the project.

Also to my parent, my mother and the memory of my father as they were always encouraging me to complete this. And to all my brothers and sisters specially my brother Maged for his support and also to every member of El-Nakla family.

Abstract

Mechatronics requires the integration of a range of technologies from mechanical engineering, electronics and software, often with conflicting requirements. This problem is particularly complex with respect to the electronic parts of a mechatronics system where there are currently no computer-based tools available to support and assist designers during the early stages of the design process.

Recognising this, the thesis was initially defined and developed based on the decomposition of a mechatronic system into its different system components using Functional Decomposition and top down approach. This led to the development and implementation of a computer-based system at lower level and which is used to assist in the design of electronic systems, which themselves form a part of a larger or mechatronic system.

Filter design was chosen as an exemplar for the purpose of demonstrating the approach used, which will also enable and support the reflection of the strategy onto other electronic systems. The system can be developed to the point of providing an expert system that support users who are either experienced or inexperienced in filter design. It can also provide help, assistance and advice to the user. Filter design knowledge has been acquired from related experts and sources such as texts and represented and coded appropriately. A range of different filter designs are stored in a well-designed database that the design can be retrieved from when needed using strategy based on a combination of Case Based Reasoning and rules.

The final part of the project is concerned with the definition of means by the methodology can be linked to other system domains to generate an overall design and to manage the transfer of information, including constraints and conflicts, between levels within the design process by means of a Data Dictionary.

A major aim of the demonstrator tool in addition to providing help and support to users is to achieve timesavings and enhance the cost effectiveness of the design process. Also it can ease the communication between different technology-groupings in a mechatronic design team.

What has been achieved is an increased understanding of the decomposition of the mechatronic system and of the interrelationships between the technical domains. At the lower levels the emphasis is on the electronic components of a mechatronic system, and of how the development of the chosen exemplar can be used as a basis from which the overall electronic system can be developed and linked to the mechatronic system.

Contents

List of Figures:

List of Tables:

Chapter 1

Introduction

The work reported in this thesis is concerned with the development of design tools and methods to assist in the design of a mechatronic system. These are based on functional decomposition and the development of an expert system as a support tool, initially for the electronic domain. It is however an important consideration of the research that the resulting system should not only support the design of the electronic element, but should also manage the transfer of information, including constraints and conflicts, between levels within the mechatronic design process.

The background of work upon which the thesis is built is set out in the previous PhD thesis "*The High Level Design of Electronic Systems*" by Richard Walters[1] and summarised in the paper "*A conceptual study for a computer-based tool to support electronics design in a mechatronic environment*"[2]. The key points arising from this prior study in relation to the work of this thesis are presented in the course of this Chapter. Although the Electronic System Design Environment was developed by Walters at the conceptual level only, and tested manually through the use of case studies, the need for a software-based approach was established in order to prove the approach and to define in detail the underlying methodology.

The work reported in this thesis extends this earlier work by first researching and defining the means by which such an environment can be realised and represented at the top-levels of the mechatronics system and then by developing and implementing a design support tool as a system demonstrator at the lower levels. For the purpose of illustrating and identifying the methods and techniques adopted, the prototype system developed in the course of the thesis uses filter design as an exemplar of the design process in the electronic domain. The resulting system is aimed at individuals who are either experts or novices in mechatronics design.

The research is also concerned with the definition of ways by which the filter design tool, and other similar tools in the electronic and mechanical domains, can be integrated to generate a comprehensive set of design tools, which will support the design and definition of the overall system.

1.1 Mechatronics Background

1.2

In discussing the background to and evolution of mechatronics as an engineering discipline concerned with the design, development and operation of a wide range of complex systems Bradley et al.[3] state that:

> "*Mechatronics provides both a title and a focus for the design and development of a wide range of engineering systems, both products and processes, in which the technologies of electronics, software engineering and information systems are integrated with mechanical engineering. The resulting combination of what have until relatively recently been considered as separate, and often competing, disciplines and the associated transfer of functionality between solution domains has been responsible for the appearance of systems in which the previous, essentially mechanical, solutions have been replaced by ones based on the integration of electronics and software with mechanical functions.*"

In relation to the link between mechatronics and electronics, Bradley & Dorey[4] go on to say that:

> "*The driving force behind the mechatronics concept in design is the ability to transfer complexity from the mechanical domain to the electronic and software domains to achieve higher performance systems at reduced real costs. Indeed, historically, each of the major technical disciplines involved in a mechatronics system has been associated with its own individual approach to design. In the case of electronics, the driving force has been the need to support the analysis and production of increasingly complex circuits on-chip.*"

Other authors have also considered the impact that mechatronics has had on the design process and Shetty and Kolk[5] define mechatronics as:

> " ..... *a methodology used for the optimal design of electromechanical products. A methodology is a collection of practices, procedures, and rules used by those who work in a particular branch of knowledge, or discipline. The mechatronics system is multi-disciplinary, embodying four fundamental disciplines: electrical, mechanical, computer science and information technology. The mechatronics design methodology is based on a concurrent, instead of sequential, approach to discipline design, resulting in products with more synergy*".

Bishop[6] meanwhile defines mechatronics as the:

> "*.... synergetic integration of mechanical engineering with electronics and intelligent computer control in the design and manufacturing of industrial products and processes*"

In considering the technical development of mechatronics suggested in the above definitions it is therefore important to note that the general structure and context of mechatronics and mechatronic systems may be represented by Fig 1.1, in which the system is separated into an energetic and information domain together with a world interface, can exist at a variety of levels.

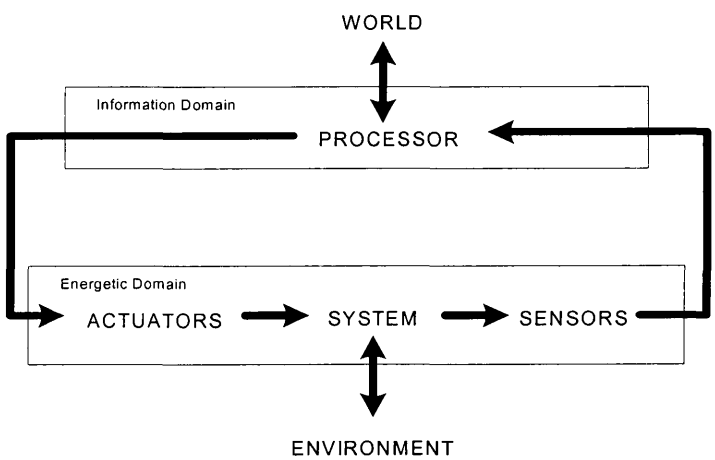


Figure 1.1: *A generalised mechatronic system*

In the context of mechatronics environment such as that shown schematically in Fig. 1.1 the designer is faced with task of partitioning the system solution between various domains as suggested in Fig. 1.2. A designer working in any of the domains and its subsequent lower levels is therefore faced with a range of decisions, which may impact upon decisions made in respect of other domains. The allocation of system functions to their domains can be achieved by different means. The one adopted here is the use of functional decomposition based on Data Flow Diagrams as will be considered later in the Chapter.

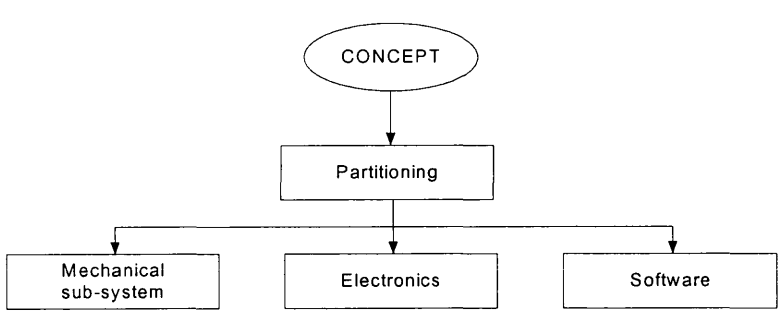


Figure 1.2: *System partitioning*

Mechatronics is widely adopted and perhaps the most readily noticeable applications are to be found in familiar, mass-produced consumer products. Microprocessor-based controllers are particularly prevalent as an integral component in modern hi-fi and video recorder systems, in automatic washing machines, in automatic cameras, photocopiers and sewing machines. Other examples of product development involving microprocessor-based controllers include a road speed limiter for heavy goods vehicles, an automatic assembly machine for the handles of woven polymer sacks and a study on the integration of expert system technology within the control system of automated banking machines[7].

1.1.1 The Design Process

The conventional electromechanical system design approach attempted to inject more reliability and performance into the mechanical part of the system during the development stage. The control part of the system was then designed and added to provide additional performance or reliability, and also to correct undetected errors in the basic design. Because the design steps generally occurred in sequence, this traditional approach is often referred to as sequential engineering.

A major source of problems arising from this approach in relation to mechatronic products lies with the inherently complex nature of a multidisciplinary system and of managing the conflicts and trade-offs between the technical domains within such a system. The mechatronics approach to system design offers a solution to this problem by applying a concurrent engineering instead of a sequential approach. In doing so it relies heavily on a computer-based interdisciplinary interaction between engineering and other disciplines[8].

Bolton[9] considered that the design process of a mechatronics system has a number of stages as follows:

1. *Need* - The design process begins with the establishment of a need from, perhaps, a customer or client. This may be supported by market research being used to define the needs of potential customers.
2. *Analysis of the problem* - The first stage in developing a design is to establish the true nature of the problem. This is an important requirement as not defining the problem accurately can lead to time wasted on designs that will not fulfil the need.
3. *Preparation of a specification* - Following the analysis of the problem, a specification of the system requirements can be prepared. This will state the problem, any constraints placed on the solution, and the criteria, which may be used to judge the quality of the design.

4. *Generation of possible solutions* - This is often termed the *conceptual stage.* Outline solutions are prepared which are worked out in sufficient detail to indicate the means of obtaining each of the required functions.
5. *Selections of a suitable solution* - The various solutions are evaluated and the most suitable one selected.
6. *Produce a detail design* - The detail of the selected design has now to be worked out. This might require the production of prototypes or mock-ups in order to determine the optimum details of a design.
7. *Produce working drawings* - The selected design is then translated into working drawings, circuit diagrams, etc. so that the item can be made.

1.1.2 Mechatronic Systems

In a mechatronic system, a computer is interfaced to signal sources and power transmission components by means of appropriate signal conditioning elements. Normally, the signal sources components are sensors and transducers, which form part of a data acquisition system, while power transmission components are actuators receiving command signals from the processor.

These conditions are illustrated in Fig. 1.3 in which the thin arrows and lines correspond to signal flow while the broad arrows correspond to power flow.

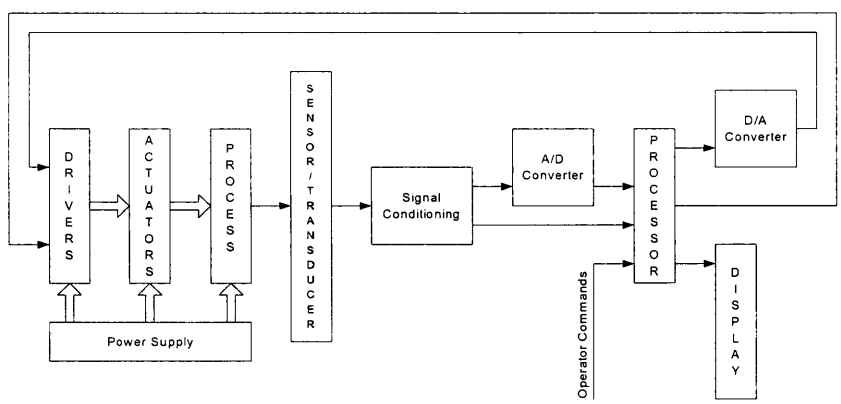


Figure 1.3: *Schematic diagram of computer based monitoring and control of a process*

Referring to Fig. 1.3, the process variables, as measured by the sensors/transducers, are signal conditioned and converted and transmitted to the processor. The processor performs real-time monitoring and control as well as signal analysis and provides the actuator commands and the necessary system monitoring and display.

The successful design of such a system requires the effective combination of the designs for the individual system components and sub-assemblies of Fig. 1.3 in order to form the overall system. The designer must carefully consider the compatibility between the individual components when proceeding with the design. For example, in Fig 1.3, it may be required to interface the analogue output from a transducer, for instance in the range 0 to 100 mV, to the analogue to digital converter that may have an input voltage in the range 0 to 5 volts. The designer should realise that part of the signal conditioning must be in the form of an amplifier with a gain of 50 to position the signal in the required range. The design of any element of the mechatronic system of Fig 1.3 is thus constrained by the inputs to, and the outputs from it, to and from other system components.

1.1.3 Functional Decomposition And Data Dictionary in Mechatronics Design

As has been indicated by Walters et al [10], a functional decomposition process may be used in the design of a mechatronic system and its components. This decomposition provides the designer with the ability to breakdown the system specification into its functional elements and to proceed through successive layers of decomposition until the system cannot be further decomposed, indicating a possible implementation for a defined process. The external communications between the processes are defined in the Data Dictionary and the process specifications.

In order to understand this, the diagram of Fig. 1.3 can be represented by a Data Flow Diagram (DFD) formed from the implementable function components as shown in Fig. 1.4. The Data Dictionary and the process specification are used when implementing any of the processes where the designer is required to define each flow specified in the data flow diagram in a Data Dictionary. These flows then represent the interface between the different system elements.

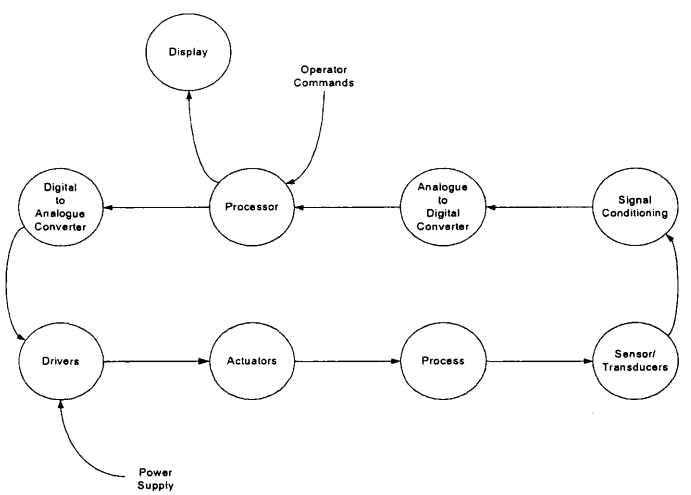


Figure 1.4: *Data Flow Diagram of computer based monitoring and control of a process*

The Data Dictionary can be used not only to indicate the interconnection between processes, but, as will be shown in Chapter 6, as a means of constraint transfer between processes[2].

1.2 Earlier Work

In the course of Walters'[11] work, two main proposals were investigated. The first of these was for the development of a unified design environment referred to as the Electronic System Design Environment (ESDE), within which the designer could work through the entire design process. The second, and arguably more fundamental proposal, was for a major shift in the ethos of the use of computers within the design process, from that of a subordinate system to which the designer turns for the answers to a specific problem or support for a defined task, to one in which its specific abilities could be constructively married with those of the human designer to generate an optimum design team within which the computer plays an integral part. Walters' work established the requirements for, and presented a discussion of, the operation of the ESDE. However, because of the scope of the project, he was unable to generate or define the relevant structures or code that would lead to system implementation.

Consider the design process itself. As French[12] suggests, this can be considered at different levels ranging from the highest level where the design is considered in very abstract terms taking into account only the functionality of the system, to those associated with detailing which occupy the final stages of the design process. In the case of high level ECAD and CASE tools, these latter stages are generally aimed at the structured development either of hardware in the form of integrated circuits, or software. The output is some form of structured language be that VHDL for hardware development or C for software development. Lower level tools are more often application based or vendor specific and are usually highly tuned to the task for which they are designed. Because of the wide and varied range of target applications; for instance available hardware forms include Application Specific Integrated Circuits (ASICs), Field Programmable Gate Arrays (FPGA), and Programmable Logic Devices (PLDs), these different tools may be quite different in their look and operation. If appropriate tools already exist, is it sensible to consider the development of an environment, which can bridge the gap between the two groups, mentioned above. Or is it better or more efficient to develop a completely separate environment, which can undertake all the tasks within itself?

Walters also argued that if commercially available tools are used for the initial input stages of the design environment, then that could result in some specific problems. Firstly, the output format of the tools may not be suitable for all parts of the system. Secondly, as the different parts of the system are developed, it may be found that there is a more suitable way

of realising the overall system. This will require the reverse transfer of information across the interface between tools, which may not be possible in the desired form. Furthermore, it may be the case that information may be required which the tool is not designed to supply. This would require the development of interfaces to process this information, and could lead to the environment becoming inefficient or cumbersome.

Walters therefore argued for the development of the entire high-level front end of the environment independently of any other tools. This allows the freedom to generate the most efficient structure for the environment itself, then, once it is fully developed, to reassess the currently available commercial tools to establish if any can satisfy the requirements of the environment.

It was also proposed that the environment should sit at a high-level within the overall design hierarchy and take its input directly in the form of the functional requirements of the system. It was further considered desirable for the system to be able to be used for all stages of the design processes, such that the output could be used to generate the appropriate hardware and software. Finally, the environment should be able to monitor the cost of the design as it develops, enabling different alternatives to be compared.

The starting point for the electronics design environment as proposed by Walters is the functional decomposition block in Fig. 1.5, with the design information input then being generated elsewhere within the design hierarchy and including links to other aspects of the mechatronic design process. Within this process, all the functional aspects of the design are considered, together with any related physical aspects such as the housing of the completed design.

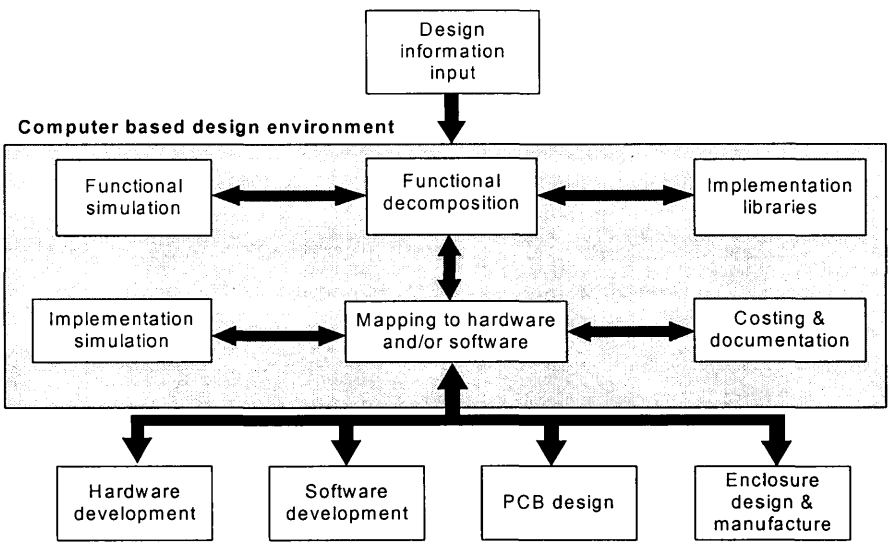


Figure 1.5: *Structuring the computer-based design environment as proposed by Walters*

A functional simulator has been included in Fig 1.5 to allow for the simulation of the design before performing any hardware mapping, confirming that the functionality of the system is correct. A link is also included from implementation libraries, enabling known implementations to be recognised. Once the decomposition phase is completed, the mapping process is begun, with the functions being mapped to their hardware or software counterparts. A simulator is included to test the results of the mapping process, and to check for any implementation errors. The simulator might ultimately be designed to account for the external environment as well as the electronic design, allowing a more complete simulation of the design to be undertaken.

Once the implementation is complete, the final level of the design environment consists of the specific tools relating to the desired form of implementation, and which are used to convert the output to the required form for the end process used. This level also provides the costing and documentation to the environment.

Working from the above, Walters identified a possible route by which an implementation could be achieved based on a process-oriented strategy as follows:

1. A function is identified from a list supplied by the ESDE at the process identification stage and this is used to identify the forms of implementation available. This information is then stored by the system until the process is to be implemented.
2. When a process is selected for implementation, the process specification is accessed to establish the necessary functionality. The appropriate implementation route is then accessed.
3. The data dictionary and process specification entries are accessed together with the associated constraint. Global constraints are also referenced and any additional information identified as necessary to the implementation requested from the designer.
4. The designer is asked to specify the optimisation criteria to be used together with the degree of fit required.
5. The implementation database is then searched for those implementations, which achieve the specified degree of fit taking into account all imposed constraints.
6. The available implementations are then presented to the designer who will be required to make a choice. Constraints associated with this choice will then be propagated to other processes as appropriate.

This procedure is illustrated diagrammatically in Fig. 1.6.

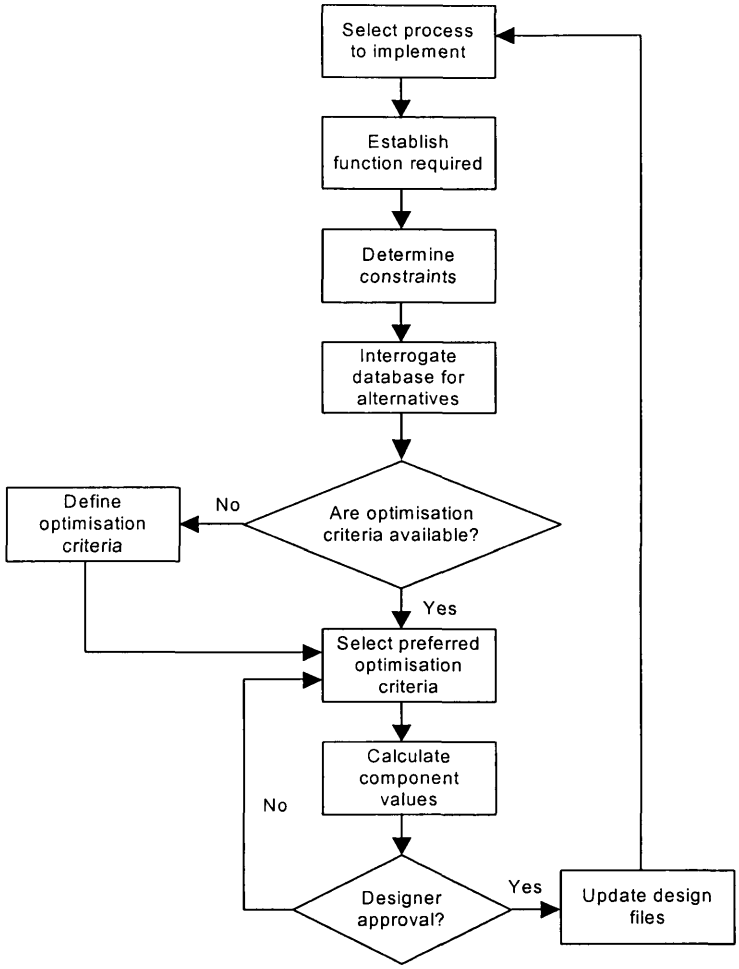


Figure 1.6: *Implementation process flowchart*[13]

Previous work therefore developed the concept of the ESDE and an outline proposal as to how such a system could be achieved. It also demonstrated how such a system might work by manually testing design case examples. It did not however attempt to investigate the mechanisms by which the implementation of the system is to be achieved, nor the requirements for the handling, management and interpretation of the associated data and knowledge.

1.3 Research Aims and Objectives

The primary aim of the research reported in the thesis is to investigate the viability of such an environment by establishing the means by which the system can be implemented. Considering the time and the size of the overall project, the work concentrated in the first instance at the stage after the completion of the functional decomposition and function simulation at the top-level of the mechatronics system in the ESDE as in Fig 1.5. The system

implementation, including the storage of the designs and the mapping to hardware and/or software at the subsequent lower levels will thus be established.

This involves research into the design and development processes at these lower levels, leading to the implementation of an expert system strategy based on a human expert's knowledge of electronic design as well as information from other sources such as texts and papers. As mechatronics is a multidisciplinary technology, the system is designed to be used by individuals who are either novices or experts in electronic design.

The system developed in the course of the research may be used to provide either a quick design route towards a possible design solution, which might suit a novice user, or offer other design routes, represented by a custom design, which might suit expert users. This means that any member of the mechatronics design team can use the system irrespective of their level of experience or knowledge. Consequently, the system will act to ease the communication between the different areas of technological expertise in the mechatronics design team.

Another key area with which the research is concerned is that of how to implement a database of existing designs as an integral part of the system. In the chosen solution, existing designs are documented in the form of a unified template and stored as design cases from which the required design can be retrieved when needed through the interaction of the user with the expert system. This supports design reuse and will also save the user time by removing the need to start each design task from scratch.

A further important issue in the research is that of deciding on the best and most appropriate reasoning techniques to be used when developing the form of expert system used to find a design fit. Traditional rule-based techniques and case-based reasoning are investigated and evaluated to determine the best strategy, or combination of strategies, to be used to embed the necessary expertise. Research was also carried out into the user-defined optimisation criteria for each of the implementations, as it is by these that the retrieved design will be judged.

In order to meet the time scale of the project, a representative electronic system had to be chosen as an exemplar for the thesis, while enabling the reflection of the strategy onto other electronic systems and ultimately to the entire mechatronic system. After consideration it was decided to develop a filter design expert system as the key demonstrator, this choice is discussed in detail in Chapter 3. Figure 1.7 shows where the proposed implementable design tool is positioned in relation to the overall mechatronics structure and to related system domains.

Links are defined between the system design hierarchy structures in Fig 1.7 to the design environment of Fig. 1.5. The box " Design information input" in Fig 1.5 is represented by the specification of mechatronic system at the very top-level of Fig 1.7. "The functional decomposition" box in Fig. 1.5 represents the mechatronic system and its decomposed components; the electronics, electromechanical and software systems at the top-level of Fig 1.7. The box "Mapping to hardware and/or software" of Fig. 1.5 and its surrounding components such as costing, hardware and software development, PCB design and so forth, are represented by the implementing one of the electronic sub-systems in Fig 1.7. This involves the development and implementation of a user assistant design tool at this lower level, as well as defining the design impact on the other levels in the mechatronic system as indicated by upwards arrows in Fig 1.7.

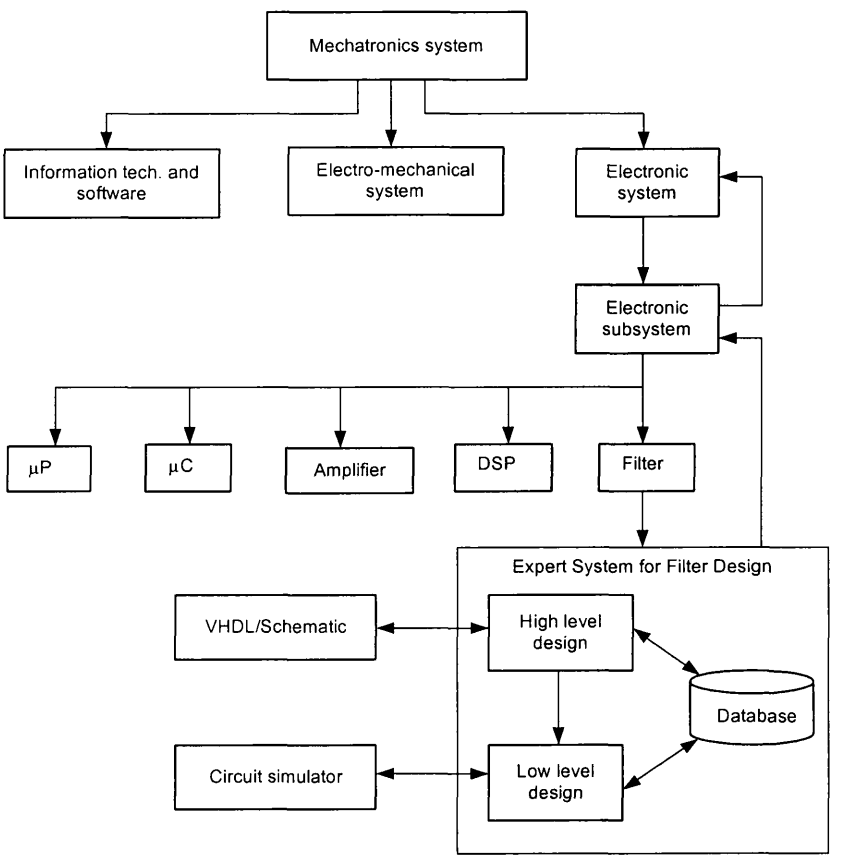


Figure 1.7: *A hierarchy of electronic design modules and mechatronics*

The thesis therefore reports initially on the development and implementation of an expert system based strategy for the design of the electronic component of a mechatronic system that will interact with and support users in developing a design and which will provide help and advice to guide users towards a design solution. Integral to this is the ability of users to

search for an existing design which may meet their criteria or to enlist the help of the system design tools, supported by techniques such as case based reasoning, to modify an existing design into one closer to their needs.

It then considers how the approach can be adapted to other electronic domains at the same level, and ultimately to the mechanical and IT domains in the upper levels of Fig.1.7. This involved assessing the impact and management of constraints and of the associated conflict resolution strategies required.

1.3.1 Defining the Boundaries of the Project – Project Scope

Having investigated the prior work on which the thesis is based, it was recognised that the implementation of such a wide ranging electronic system design tool was a considerable task, the overall magnitude of which was considered to be beyond the possibilities of a project of this duration. Therefore, the project focussed on the question of developing an interactive system to assist the user at the lower level detailed design stage. Having completed the system demonstrator, it was then extended to show how the approach can be linked to other areas of the design process at the same and other levels.

There were several aspects of the demonstrator tool identified as being of particular interest and importance to the project. Firstly, the implementation of such a tool will open the way to further research towards the implementation of the rest of the environment and the combination of all parts to create an implemented, working environment.

The second benefit to be gained will be that of moving the proposed electronic system design environment from the conceptual stage of development into an implementation phase. The expert system strategies adopted for the tool mean that current designs can be reflected in future design studies, supporting design re-use and preventing repetition or unnecessary redesign to support time saving and cost effectiveness. The final benefit is that it will support the understanding and definition of the strategies necessary to link different systems at different levels, in which any change in one system will flag that change to the others appropriately.

1.3.2 Research Aims

The main aims and activities to be included in the research are:

- The investigation of the generic structure of the required expert system based on a consideration of user requirements. This will include the acquisition of filter design knowledge by contacting relevant human experts to discuss design scenarios as well

as through the use of texts and other references. The knowledge obtained from this process will be analysed and used in the structuring of the filter design process.

- The encoding of the knowledge into an appropriate expert system form through the knowledge representation process. With an increasing number of design cases, there is a need to document these cases in a database. The system would then be able to retrieve data from the database as well as to store new data in the form of new designs. This will then lead to the production of a unified filter design template capable of capturing different filter designs.
- The definition of all aspects of the required expert system, including the knowledge base and case base. These are then to be incorporated as part of the design process. User defined designs and optimisation criteria are examined using fuzzy logic as compared to the use of crisp values as part of the selection process.
- The implementation of a prototype system based around active filter technology. This will include building the physical database file and the appropriate code to retrieve the design from the database.
- The testing of the system and its associated interfaces to ensure that they perform according to specification and meet user expectations. Also to ensure that the implemented system represents the intended interactive tool.
- Use the results of the implemented filter design expert system as a means of defining and implementing other systems in the electronic domain and within the overall mechatronics structure. Means of linking such a system to other electronic domains and also to other mechatronics domains will also be considered.
- Identify areas requiring further work, enabling a statement of requirements for future work to be developed.

1.4 Thesis Structure

The remainder of the thesis is structured as follows. Chapter 2 considers the different aspects of electronic design and expert system beginning with the historical context of electronic design, it then looks at the use of computers in the design process and examines the different ways in which computers can be used to aid the designer. Consideration is then given to the use of Electronic Computer Aided Design (ECAD) tools from circuit level to the most complex application at system level. Later parts of Chapter 2 are devoted to the presentation and definition of the expert system architecture and the associated system development

process. The Case Based Reasoning (CBR) approach is considered along with the applications of expert systems in electronic design.

Chapters 3, 4 and 5 represent the bulk of the work undertaken in the project, which itself may be divided into three main sections. The first of these, discussed in Chapter 3, was to investigate the structures for an expert system to support filter design, with the approach eventually selected based on the Case Based Reasoning (CBR) technique. The final part of the Chapter deals with fuzzy reasoning and how this may be used in setting the optimising criteria.

Chapter 4 considers the modelling of the system using formal methods as part of the design and implementation process to establish the software and related structures. The first part of this Chapter examines some of the software methods available for system development with a particular emphasis on process oriented and object oriented methods. Also in this Chapter Data Flow Diagrams (DFDs) are used to generate a process model, which represents the system from its highest level down in increasing detail. The second part of the Chapter is concerned with the detailed design processes for the system, involving the design of the user interface, system dialogues and files and databases, including all fields and records.

Chapter 5 describes the implementation of the system in the form of working and reliable software. The first part of this Chapter focuses on the database design and the building of the physical database file, including the necessary data access management and data control functions. Because of the similarities in the basic design principals across filter types it was decided concentrate on active filters in the first instance. Once a robust system has been developed for active filters, the prototype can be expanded to handle all filter types, as well as the associated database file.

The second part of Chapter 5 is concerned with the testing of the application to ensure that it performs according to specification. Finally, a usability test is conducted which tests how convenient the system is to use.

Chapter 6 then examines how similar techniques to those used in the developed expert system may be used to produce a similar system for other electronic sub-systems. It also looks at ways of linking an individual system to other domain systems and to the entirety of the mechatronics system. The first section of the Chapter therefore defines the basis on which the different mechatronics components are interfaced while the second part of the Chapter examines the Data Dictionary and its role in defining the interconnection between, and as a means of constraint transfer between, processes. This is demonstrated by choosing an appropriate design example of a mechatronic system.

Chapter 7 provides the project conclusions and discusses the issues raised during its progression, and the problems encountered. A list of the major contributions is also presented. The Chapter also considers the future development of the research, setting out possible routes of progression and highlighting those areas, which are considered to require the greatest attention.

Finally, Appendix A contains a complete description of the filter design exemplar, including the full functional decomposition and the implementation code. Appendix B contains the formulas for a range of filter types while Appendix C contains research publications.

---

[1] Walters, R. M. (1996), *The high-level design of electronic systems.* PhD thesis. Lancaster University.

[2] Walters, R. M., Bradley, D. and Dorey, A. (2000). A conceptual study for a computer-based tool to support electronics design in a mechatronic environment. *Microprocessors and Microcomputers,* 24: pp. 51- 61

[3] Bradley, D., et al. (2000). *Mechatronics and the design of intelligent machines and systems*. UK: Stanley Thornes.

[4] Bradley, D. and Dorey, A. P. (1994). Measurement science and technology- essential fundamentals of mechatronics (review article), *Measurement Science and Technology,* (5): pp.1415-1428

[5] Shetty, D. and Kolk, R. A. (1997). *Mechatronics system design*. USA: PWS publishing company.

[6] Bishop, R. H. ed. (2002). *The Mechatronics Handbook*. USA: CRC Press LLC.

[7] Fraser C. and Milne, J. (1994). *Integrated electrical and electronic engineering for mechanical engineers*. UK: McGraw-Hill international

[8] op. cit., 5

[9] Bolton, W. (1995). *Mechatronics: electronic control systems in mechanical engineering*. UK: Addison Wesley Longman Limited.

[10] op. cit., 2

[11] op. cit., 1

[12] French M. J. (1985). *Conceptual Design for Engineers*. 2nd edition. UK: Design Council Publications.

[13] op. cit., 1

Chapter 2

Electronics Design and Expert Systems

2.1 Introduction

Mechatronics is a multidisciplinary engineering system, the driving force behind which is the ability to transfer complexity from the mechanical domain to the electronic and software domains, achieving higher performance systems at reducing cost. In the case of electronics, the driving force has been the need to support the analysis and production of increasingly complex circuits on chip[1].

The initial focus here is on the development and implementation of a computer based design tool for electronic design to support and assist the design team during the early, decision making, stages of the mechatronics design process. To place this into context it is necessary to briefly review the evolution of electronics design with an emphasis on the development of the design tools and simulation packages. Later sections of the Chapter consider the role of an expert system approach in comparison to more conventional methods and define the expert system architecture used and the associated system development. The case based reasoning approach (CBR) and its development are explored and finally the Chapter considers the application of expert systems in electronics design.

2.2 The Evolution of Electronic Design

Over the last 50 years the electronics industry has grown rapidly in both scale and complexity. The development from valves to transistors and circuits with relatively few discrete components to complex integrated circuits with a million or more transistors on a single chip has been supported by the development of a range of Electronic Computer Aided Design (ECAD) tools. Such tools are now used extensively in designing Very Large Scale Integration (VLSI) systems such as microprocessors, micro-controllers and ASICs.

Electronics technology and systems developed rapidly in the latter half of the 20th Century from relatively simple circuits designed with pen and paper to extremely complex integrated circuits developed using ECAD or Electronic Design Automation (EDA) tools. A brief timeline of standard (off the shelf) computer integrated circuits over the last three decades is shown in Table 2.1.

Table 2.1: *Integrated circuit development*[2,3]

Year	Event
1971	First microprocessor, Intel 4004, 4bit, 2300 transistors (10 microns), 108 kHz clock speed. EPROM.
1972	Digital Signal Processor (DSP) invented. Intel 8008, 8 bit, 3500 transistors (10 microns), 200kHz clock speed.
1974	Intel 8080, 8 bit, 6000 transistors (6 microns), 2 MHz clock speed. 4 Kbit DRAM.
1976	16 Kbit DRAM
1978	Intel 8086, 16 bit, 29000 transistors (3 microns), 5-10 MHz clock speed. Intel 8088, 8 bit, 29000 transistors (3 microns), 5-8 MHz clock speed.
1979	64 Kbit DRAM.
1980	Modern DSP commercially available.
1982	Intel 80286, 16 bit, 134000 transistors (1.5 microns), 6-12.5 MHz clock speed. 256 Kbit DRAM.
1983	First CMOS DRAM. EEPROM.
1984	Flash memory.
1985	Intel 80386 DX, 32 bit, 275 transistors (1 micron), 16-33 MHz clock speed
1986	1 Mbit DRAM
1989	Intel 80486 DX, 32 bit, 1.2 million transistors (1 micron), 25-50 MHz clock speed.
1991	16 Mbit DRAM
1993	Pentium II processor, 64 bits, 3.1 million transistors (0.8 micron), 60-66 MHz clock speed.
1994	64 Mbit DRAM
1995	Pentium Pro processor, 5.5 million transistors (0.35 micron), 150-200MHz-clock speed.
1997	Pentium II processor, 7.5 million transistors (0.35 micron), 200-300 MHz clock speed.
1998	256 Mbit DRAM
1999	Pentium III processor, 9.5-28 million transistors (0.25-0.18 micron), 450-733 MHz clock speed.
2000	Pentium 4 processor, 42 million transistors (0.18 micron), 1.4-1.5 GHz clock speed.
2002	Pentium 4 processor, 55 million transistors (0.13 micron), 2.2-3.06 GHz clock speed.

The developments set out in Table 2.1 were supported by the introduction of ECAD tools, supporting both the circuit designer and the designer of the integrated circuits themselves. The power of these tools was then enhanced by the availability of the faster computers that the tools were being used to design. Indeed, the development of ECAD tools and of computer hardware are each based on developments in the other field. Thus, with the appropriate software, a new chip design can be created, simulated, tested and then manufactured. This new chip is then used as the basis for a next generation computer, which will incorporate new features and speed and the specifications of which can be used to develop and run new ECAD software with new capabilities.

VLSI chip development has been paralleled by major advances in packaging and mounting and packages are available with pin counts approaching 500. Many packages are designed for surface mounting, saving space and increasing packing density by permitting chips to be placed on both sides of a printed circuit board (PCB). Another step in the drive to reduce space is the Multi-Chip Module (MCM), employed not only for hybrid ICs but also to ease the interconnections of VLSI chips

ECAD tools can be used not just for computer chip design or standard ICs, but also for custom ICs and ASICs produced to a customer's specification by the vendor. ASICs were

introduced around 1980[4] and are classified into full custom and semi-custom forms, the latter being mask based and cheaper for mass production.

ASICs are specialised circuit blocks or entire chips which are designed specifically for and dedicated to a given application or application domain. For instance, a video decoder circuit may be implemented as an ASIC chip to be used inside a personal computer product or in a range of multimedia devices. Due to the custom nature of these designs, it is often possible to squeeze in more functionality under the performance requirements while reducing system size, power, heat, and cost than is possible with standard IC designs. As a result of cost and performance advantages, ASICs and semiconductor chips with ASIC blocks are used in a wide range of products, from consumer electronics to space applications[5]

One particular form of semi-custom ASIC is the programmable logic device (PLD) on which the desired logic functions are obtained by the programming of the masked interconnections. Another member of the ASIC family is the field programmable gate array (FPGA), first introduced in 1985 and which, because it is user-programmable, offers the prospect of quick development at greatly reduced cost, making it attractive for proving a design. Quantity production is however more expensive, because the FPGA is user, and not mask, programmed[6].

Microelectronics technology has matured considerably in the past few decades. Systems, which until the start of the 1990's required a printed circuit board for implementation, are now being developed on a single chip. These Systems on Chip (SOCs)[7] are becoming a reality as a result of major improvements in chip fabrication and process technology enabling smaller on-chip components.

In the past few years, the reuse of semiconductor functional blocks has become a significant part of the design process. High-level functional blocks such as signal processing functions, input/output interface devices, audio/video compression and decompression functions etc., are being designed once and reused in several designs. These blocks are known as *cores* and several companies specialising in developing these cores are selling them as Intellectual Property (IP)[8]. These cores are designed with clear, well-defined and documented interfaces so that they can easily be integrated into system design. The resulting system uses several of these cores, and sometimes a microprocessor core, to implement a complex system targeted at say, multimedia processing. This is akin to the use of software component libraries in software design[9].

The tasks of design, fabrication and testing of even the simplest IC involve many operations and stages. An error in any of these stages can render the final circuit inoperative, so careful checking of the design stages is therefore imperative. Many of the operations involved are

repetitive, tedious and error prone, and are therefore much more suited to computers than humans.

2.3 Computers in Electronic Design

As more and more computer based aids are developed and made available, it is sometimes difficult to judge which of the available ECAD tools it is the best to use in a particular circumstance because of the speed at which new product releases become available.

2.3.1 Hierarchical Levels

Before describing the specific computer tools, it is worth briefly discussing the various levels of abstraction at which an electronic system can be considered. This is because specific ECAD tools tend to operate at a particular level of abstraction, although application of the tools at mixed levels is becoming more common. Based on a 'top-down' approach, the levels are[10]:

System level - This may consist of one or more boards of ICs or an MCM system and its associated interconnections, forming a complete integrated system.

Chip level - This consists of a complete IC and may be considered as a packaged device, with associated parasitic elements, or a complete, unmounted circuit die.

Sub-circuit or register transfer level (RTL) - This consists of a significant piece of circuitry with a particular function. In the digital area, for example, it may be a multiplexer, adder stage, shift register, memory block, etc.; while in the analogue area it may be filter, oscillator, ADC/DAC, etc.

Gate or circuit level - This is the 'building block' level, individual gates in the digital circuits or op-amps, voltage references, switches, etc. in the analogue area.

Transistor level – This is the individual component level, mainly transistors, but also including capacitors, resistors, transmission lines, etc.

Mask or layout level - The lowest level of abstraction is the mask level, which contains the various polygonal shapes at different layers that are used in the actual IC fabrication and which when combined together, provide the exact topology of the circuit.

2.3.2 Design Entry

The purpose of design entry is to describe an electronic system to a set of EDA tools. Electronic systems used to be, and many instances still are, constructed from off-the-shelf components, such as Transistor Transistor Logic (TTL) devices[11]. Design entry for these

systems now usually consists of drawing a picture, a schematic, showing how all the components are connected together. This type of design entry process is called schematic entry, or schematic capture.

The circuit schematic is a picture, an easy format to understand and use, but computers need to work with a version of the schematic referred to as a netlist. The output of a schematic entry tool is thus a netlist file that contains a description of all the components in a design and their interconnections.

Schematic entry tools for circuit design enable components or devices to be picked from a library and assembled to represent the circuit, see Fig. 2.1. However, drawing every component for an IC is impractical without using a hierarchical approach to reduce the size and complexity of the schematic. An schematic can contain sub-schematics, each sub-schematic may then, in turn, contain other sub-schematics.

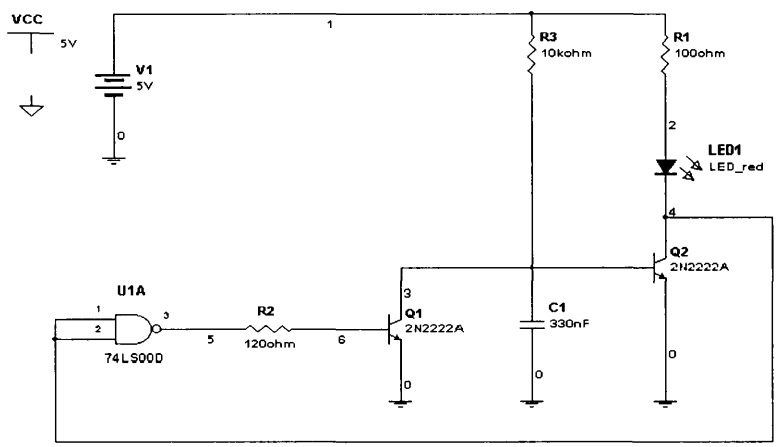


Figure 2.1: *An example of simple circuit schematic*[12]

Once a design has been created a wide range of circuit simulation packages is available by which it can be tested and evaluated, enabling any required modifications to be made before construction. The best known of these packages is probably SPICE[13]. This was originally developed by the Electronics Research Laboratory of the University of California and became widely available in 1975. SPICE, and its derivatives such as PSPICE, can be used for both analogue and digital circuits and uses a component library to store the characteristics of each device. Standard SPICE libraries include details of a range of active and passive components, including transistors and integrated circuits. Table 2.2 sets out some currently available circuit simulators[14].

Table 2.2: *Commercially available circuit simulators*

Circuit Simulator	Application	Made by
HSPICE	Analogue Circuit	Meta-Software
PSPICE	Analogue Circuit	Microsim
MICRO-CAP	Analogue Circuit	Spectrum Software
PLOGIC	Digital Circuit	Microsim
QUICKSIM	Digital Circuit	Mentor Graphics
WORKVIEW	Digital Circuit	Viewlogic
SABER	Mixed Analogue- Digital	Analogy
ICAP	Mixed Analogue- Digital	Intusoft
PSPICE A/D	Mixed Analogue- Digital	Microsim

A schematic can be a very effective way to convey design information because pictures are a powerful medium[15]. There are however two major problems with schematic entry. The first is that making changes to a schematic can be difficult, if the need is to include an extra few gates, it may be necessary to redraw the whole sheet. The second is that for many years there were no standards as to how symbols should be drawn or how the schematic information should be stored in a netlist. These problems led to the development of design entry tools based on text rather than graphics, the Hardware Description Languages (HDLs).

- Low-level design languages - These languages are useful for describing state machines and combinational logic and evolved from the design of PLDs. It is natural that designers would then want to use PLD development systems and languages to design FPGA and other ASICs. Examples of PLD programming languages are ABEL from Data I/O, CUPL from Logical Devices and PALASM from AMD/MMI[16] .

- High-level design languages - These represent the set of HDLs that includes options such as very high speed integrated circuit hardware description language (VHDL) and Verilog. These are more complex but capable of describing complete systems and are used interchangeably. VHDL was initially used for description and documentation, and then later for design entry, simulation and synthesis while Verilog is a simulation language [17,18].

Even with ECAD tools, designers still need a knowledge and background in electronic design, to which they then add the capability of the tools. If the knowledge of those experts can be represented as an expert system for users needing design support, the capability of the design team can be enhanced. This is particularly the case in mechatronics where team members such as mechanical and software engineers can gain an insight into options available at an early stage in the design process.

2.4 Background to Expert Systems

A number of texts and papers were consulted to provide background on expert systems[19,20,21,22,23,24]. These gave a range of opinions and options as to the role, purpose and definition of an expert system. Of these texts, that by Awad[25] was felt to provide a particularly well set out and structured discussion and was used as the core text.

A standard definition of an expert system is difficult to articulate. A survey of more than 80 books and 200 published papers carried out by Awad[26] apparently revealed about as many definitions as there were publications. The definition ultimately used by Awad, and also used here, is that:

> "... *an expert system consists of computer software programs that emulate or clone the reasoning of a human expert in a problem domain*[†]".

The characteristics that distinguish expert systems from conventional systems are summarised in Table 2.3.

Table 2.3: *Comparison of expert system and conventional programs*[27]

Characteristic	Expert System	Conventional System
Underlying paradigm	Heuristic, usually implemented using state space search. Solution steps implicit (i.e. not determined by programmer). Solution if found, not always guaranteed optimal or correct. Usually declarative problem-solving paradigm.	Algorithmic. Solution steps explicitly written by programmer. Correct answers given. Procedural problem-solving paradigm.
Method of operation	Reasons with symbols. For example, infers conclusions from known premises in order to diagnose a patient illness. Inference engine is used to decide the order in which premises are evaluated	Predominantly manipulates data. For example, sorting, calculating and sorting data processed to produce information such as payslips for a company payroll system
Processing unit	Knowledge. This may be represented in the form of rules. Knowledge is active in that an expert system can reason with knowledge to infer new knowledge from given data	Data. Typically represented in the form of arrays or records in language like C or COBOL.
Control mechanism	Inference engine is usually separate from domain knowledge	Algorithm + data
Fundamental components	Highly interactive. Usually takes form of question and answer session	Variable
Explanation capability	Yes. An explicit trace of the chain of steps underlying the reasoning processes. Would typically enable a user to find out how the system arrived at its conclusions.	No
Learning capability	Yes, but limited	No

[†] A problem domain is a special subject area in which the domain expert can act to solve problems.

In any comparison of expert systems with conventional programs, expert systems are seen as adapting a heuristic approach to computation as compared to the numeric based strategy of conventional computation. Expert reasoning is based on symbolic manipulation and incorporates judgement into the system and uses decision criteria in reading a solution.

In arriving at a solution, an expert system does not therefore follow a pre-programmed path; rather it proceeds on the basis of interpreting stored knowledge. This stored knowledge, which generally takes the form of rules, does not have to be held in any particular sequence as the expert system weighs facts, rules, and assumptions and makes decisions appropriate to the problem under consideration.

This flexibility is a result of declarative programming, which gives expert systems their rule sequence insensitive nature. In contrast, traditional systems use procedural programming, by which all paths through the programme are defined by the programmer.

Even though expert systems have advantages such as consistency, permanency and support for fast processing; they also have their drawbacks. Consideration of the relative merits of computers and humans by Bradley et al.[28] suggests that there is currently an almost complete reversal between those tasks to which a human is suited and those to which a computer is suited. It is not therefore suggested that the computer should be considered as a replacement for the human designer, since humans possess attributes well suited to the design process. However, as the human and computer possess different but complementary attributes, if these can be combined in a constructive manner so that the best use may be made of specific abilities of both, then a powerful and well-rounded design team should be possible. Table 2.4 compares human expertise with expert systems.

Table 2.4: *Comparison of Human Expertise and Expert Systems*[29]

Human Expertise	Expert Systems
Perishable	Permanent
Unpredictable	Consistent
Slow reproduction	Quick replication
High-priced	Affordable
Slow processing	Fast processing
Creative, adaptable, broad focus	Needs instruction
Common sense as knowledge	No common sense

2.4.1 Architecture of an Expert System

All expert systems have three components: a knowledge base, an inference engine and a user interface. Figure 2.2 illustrates the basic relationships between these core components.

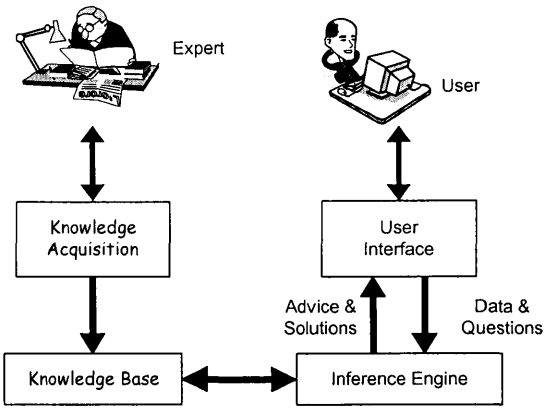


Figure 2.2: *Components of an expert system*

Knowledge Base:

The knowledge base is the repository of the rules, facts and knowledge acquired from human experts and other relevant sources. Most expert systems use production rule structures, which is why many expert systems are referred to as rule-based systems. Knowledge is typically represented by statements or rules of the form:

IF (premise) THEN (conclusion)

These rules then incorporate the relevant facts and assumptions about the problem the system is intended to solve and therefore lie at the heart of the knowledge base.

A rule that offers only one possible conclusion is easy to write. For example:

IF "*An analogue to digital converter is to be used*"
THEN "*An anti-aliasing filter is required*"

This rule could then be linked with other rules such as:

IF "*An anti-aliasing filter is to be used*"
THEN "*An analogue lowpass filter is required*"

Rules that have multiple conclusions or uncertainty and fuzziness are more difficult to write and verify [30]. One way of dealing with uncertainty is to assign a confidence factor to identify the degree of certainty associated with a given attribute. Similarly, fuzzy data have to be defuzzified before a knowledge base or inference engine can use them.

Inference Engine:

Even when it is possible to represent domain knowledge as rules, a human expert would have to know not only which rules to apply, but also in what order they should be applied to

solve a particular problem. Similarly, an expert system needs to decide which rules, and in what order, should be selected for evaluation.

To do this, an expert system uses an inference engine. This is a program that interprets the rules in the knowledge base in order to draw conclusions. Two main alternative strategies are used with rule-based systems; backward chaining and forward chaining. A particular inference engine may adopt either or both.

A backward-chaining inference engine is "goal orientated" in the sense that it tries to prove a goal or rule conclusion by confirming the truth of all of its premises[31]. Consider a lowpass elliptical filter. This is characterised by having ripples in both the passband and the stopband (see Fig 2.3†). This characterisation may then be represented by a rule of the form:

IF "*The filter response has a ripple in the passband*"
AND "*The filter response has ripples in the stopband*"
THEN "*The filter response is elliptic*"

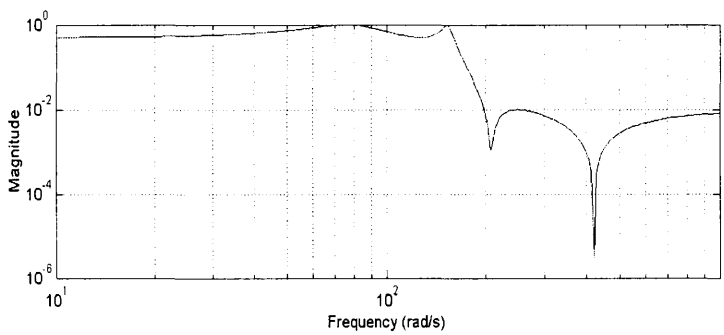


Figure 2.3: *An elliptic filter response*[32]

In contrast, a forward-chaining inference engine starts by examining the current state of the knowledge base and finds those rules whose premises can be satisfied from known or given data, and then adds the conclusions of those rules to the knowledge base. It then re-examines the complete knowledge base and repeats the process, which now progresses further since new information has been added. The earlier example can be reconstructed for forward-chaining as follows:

IF "*The filter response is elliptic*"
THEN "*The filter response has a ripple in the passband*"
AND "*The filter response has ripples in the stopband*"

† Note the frequency unit in rad/s when Matlab is used throughout the thesis.

In this project both backward and forward reasoning are used along with another reasoning technique called case-based reasoning, which will be covered in more detail in section 2.4.3.

The User Interface:

An expert system user interface is normally of a highly interactive nature to reflect the form of dialogue that takes place between a client and a human expert. The expert system user interface will not only enable the user to ask and receive answers to questions, but will also allow the user to interrupt its operation by asking for explanations and to get help and advice. Advice from the expert system might take the form of textual or graphical output depending on the nature of the expert system domain.

Many of the user interface design issues for expert systems are the same as those for conventional systems. The nature of the interaction with an expert system is however, such that the interfaces for expert systems may have additional needs as they create a different set of demands on the interface. Unlike a conventional program, an expert system is not a tool that implements a process, but rather a representation of that process. Chapter 5 will show the design of the user interface for the filter design expert system, taking into account the functionality and the usability of the system.

2.4.2 Expert System Development Life Cycle

The building of a computer-based system can be viewed as a life cycle that begins with a problem and ends in a solution in the form of a running installation. In building expert systems, knowledge engineers work with experts to identify the problem domain and plan the development strategy and process from beginning to end, while in conventional systems, the system analyst gathers information from the user who has no ready solution. The flowchart of the expert system development life cycle is shown in Fig. 2.4.

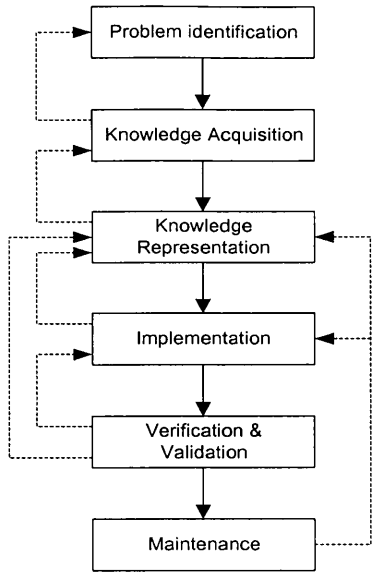


Figure 2.4: *Expert system development life cycle*[33]

Conventional system development is primarily sequential; that is, particular steps are carried out in a particular order. Design cannot be initiated without analysis, testing cannot be done without a design, and so on. In contrast, expert system development life cycle is incremental and interactive.

An expert system is not therefore built in a few large steps; rather, it evolves toward a final form. This iterative approach to development, referred to as prototyping, is by far the most successful paradigm used for expert system development.

Prototyping plays a major role in expert system evolution. For instance, to build a filter design expert system, the developer might identify only one goal and a small part of the domain as a starting point, for instance lowpass filter design. This is then built as a prototype and then the other types of filter such as highpass, bandpass and bandstop are added. The first prototype gives an indication of the likely look and feel of the end product and may well serve as a basis for the future development. Further refinements may involve adding manageable chunks to the existing prototype. This is referred to as incremental prototyping since the reminder of the development will proceed with incremental advances on the first prototype. Fig.2.5 illustrates the incremental prototyping approach to expert system development.

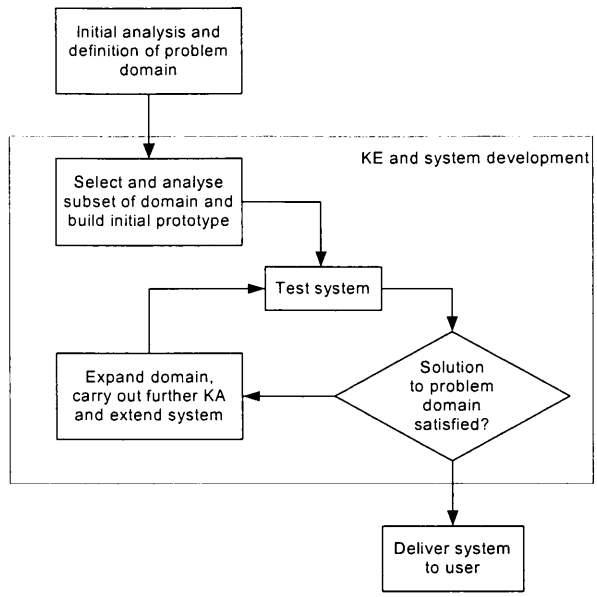


Figure 2.5: *Incremental expert system prototyping*[34]

Problem Identification:

A problem domain is the special problem area in which an expert operates, and the first step in building an expert system is the identification of a unique problem domain that is potentially solvable with an expert system. Expert systems, unlike human experts, are generally designed to be experts in a single problem domain. In assessing whether a problem is suited to an expert system approach, a feasibility study can be undertaken in which the problem is evaluated against selected criteria. However, a feasibilty assessment strategy is necessary. One such strategy, due to Beckman[35], derives a list of issues and then assigns each issue a weight that reflects its relative importance. A score is then assigned to each issue. The total sum of these weights is calculated and used to ascribe a value to the likely success of the expert system project. The three main general criteria to consider in a feasibility study are costs, appropriateness and availability of expertise.

During the course of this project the feasibility study is also evaluated using the knowledge obtained from contacting human experts. The availability of the human experts and the design knowledge of the problem domain formed an important part in the feasibility study. The cost and the implementation of the project were also reviewed and established that all the software and hardware needed for the project were available at the University.

Knowledge acquisition:

In the second step of the expert system development life cycle, the knowledge engineer accesses the expert's knowledge. Knowledge acquisition involves eliciting, analysing, and interpreting the knowledge that a human expert uses to solve a particular problem. It corresponds to systems analysis in conventional system development.

In both cases, interviews with experts are used to gather information and knowledge. However interviewing the ultimate user of the expert system may be just as important as interviewing the expert in that knowledge from the expert is needed to build the knowledge base, while information from the user ensures that a usable system will be built.

In conventional software system development, the systems analyst gathers information from the user to define the problem and hence determines alternative solutions and their consequences. In expert system development, the knowledge engineer acquires heuristic knowledge from the expert in order to build the knowledge base that emulates the expert's thought process. The user has more to gain from the expert system than does the expert who is building it, as the expert hands over years of knowledge for a system that someone else will use.

The initial knowledge acquired in this project was gained from electronic design expertise represented by members of staff at the University of Abertay Dundee involved in electronic design. The responses received then needed to be supported by other sources of knowledge such as filter design texts and Internet sources. The knowledge acquisition process for the filter design expert system will be covered in more detail in Section 3.3.

Knowledge representation:

Once the knowledge has been acquired, the next step is that of knowledge representation, coding the knowledge in such a way that an inferencing program will be able to use stored knowledge whenever needed to draw conclusions.

Either rules or frames are typically used to represent the knowledge within a knowledge-based system. A rule is a natural language conditional statement that specifies an action to be taken if a certain condition exists. Such statements are often called premise-conclusion rules and have an IF...THEN format. Each rule is independent and is based on heuristics rather than an algorithm. An example of a rule of this type is that shown earlier for an anti-aliasing filter which had the form:

IF "*An analogue to digital converter is to be used*"
THEN "*An anti-aliasing filter is required*"

A knowledge base built on rules is called a rule-based system. A high level programming language or shells can be used to implement the knowledge base.

Frames, like rules, carry knowledge. They associate objects with facts, rules, or values. A frame consists of attributes, called slots. Each fact or value stored in a slot is related to specific object or thing[36]. In relation to the design of a particular system, each available design can be held as a frame.

Suppose that there are a number of design cases and each case has specific design attributes, the system can then search for the design case which most closely matches the user requirements, i.e. the technique of case based reasoning. With this approach the user searches the existing design database to establish which solution most closely matches the user requirements. If the returned design does not fully satisfy those requirements, it can then be modified to meet the requirements or used as the basis for the creation of a new design.

In order to achieve its functionality, the expert system has to integrate both rule-based and case-based techniques. These knowledge representation techniques along with designing the database and filter design template will be covered in more detail in Section 3.4.

Implementation:

Once the appropriate knowledge has been represented in the database, the next task is to implement the system on a computer. Implementation is the process of organising the knowledge and integrating it with the processing strategy for testing. The specifics of the procedure are dictated by the chosen programing language. Visual Basic and SQL have been chosen here to implement the expert system along with the Access database and a HTML help file. The implementation of the expert system will be covered in detail in Chapter 5.

Another way of looking at implementation is that it is the transformation of a precise representation of knowledge into a machine-executable computer program. The initially implemented expert system will be expected to undergo a number of modifications before it becomes a "good enough", working expert system. Versions are modified through verification and validation, which are the next steps in the expert system development life cycle.

Verification and Validation:

Verification and validation of an expert system are equivalent to the testing stages in building a conventional information system. Verification consists of putting the expert system through a procedure to ensure that the system is right - that the programs do what they are designed to do. The internal make up of the system is checked to see that rules fire

when they are supposed to fire. In this way the technical performance of the system is evaluated[37].

Validation involves testing the system to ensure it is the right system - that it meets the expert's expectations. Validation provides assurance that the solutions or advice derived from knowledge base come close enough to those of the human expert. In other words, the vaildation process checks reliability of the expert system. Validation and verification procedures for the filter design expert system will be discussed in Chapter 5.

Maintenance:
The final stage in expert system development life cycle is that of maintenance, which ensures that the system continues to function according to the initial standards of performance. One element of this stage is preserving and modifying the knowledge stored in the system.

Expert systems must be upgraded over time to expand or improve the quality of the accuracy and usefulness of their advice. Therefore, periodic evaulation and upgrade are essential. The knowledge engineer may want to make changes, perhaps to modify the system to make rules work in a particular context, or make some other change to suite the user needs. Changes could also be possible when new knowledge has been acquired. Other changes may be necessary as result of upgrading the operating system to a newer version.

2.4.3 Case Based Reasoning

To build an expert system, there is a need for an expert and a knowledge engineer. The knowledge engineer tries to discover the rules that the expert uses to solve problems in the given domain. A typical expert system may comprise several hundred such rules. The problem is that it is often extremely difficult for experts to express their knowledge as rules. However, a human expert generally finds it easy to recount episodes about specific cases. This suggests that humans do not in fact encode knowledge as rules, but as cases. Furthermore, it should be easier to extract knowledge from an informant as cases, rather than as rules. Thus, the knowledge acquisition bottleneck for the expert system could be allayed if the programs could assimilate cases, rather than requiring explicit rules.[38]

Case based reasoning was introduced to the community by Kolodner[39] and Schank[40] and the basic problem solving model of case based reasoning grew out of projects at Yale University. Over the last twenty years, case-based reasoning (CBR) has grown from a rather specific and isolated research area to a field of widespread interest. Activities are rapidly growing as seen by the increased rate of research papers, availability of commercial products, and reports on applications in regular use. A number of texts and papers were

reviewed to provide a background on cased based reasoning, among these is one by Janet Kolodner[41] who stated that:

> *"Case based reasoning means using old experiences to understand and solve new problems. In case based reasoning, a reasoner remembers a previous situation similar to the current one and uses that to solve the new problem. Case based reasoning can mean adapting old solutions to meet new demands; using old cases to explain new situations; using old cases to critique new solutions; or reasoning from precedents to interpret a new situation (much like lawyers do) or create an equitable solution to a new problem (much like labour mediators do)."*

Riesbeck and Schank[42] defined case based reasoning in a contrast to rule-based reasoning as:

> *"Case based is an alternative to rule-based reasoning. A rule-based system will be flexible and produce nearly optimal answers, but it will be slow and prone to error. A case based system will be restricted to variations on known situations and produce approximate answers, but it will be quick and its answers will be grounded in actual experience. Case based reasoning offers two main advantages over rule-based reasoning. First, expertise is more like a library of past experience than a set of rules; hence cases better support knowledge transfer and explanation. Second, any real world domains are so complex that it is either impossible or impractical to specify fully all the rules involved; on the other hand, cases, i.e., solutions for problems, can always be given."*

The basic idea of case based reasoning is therefore to take solutions that were used to solve old problems and adapt them to solve new problems. CBR is therefore a method of inference fundamentally different from other approaches. Instead of relying on a general knowledge of a problem domain, or making associations between problem premises and conclusions, CBR is able to utilise the specific knowledge of previously experienced, concrete problem situations referred to as cases.

A new problem is solved by finding a similar past case and reusing it in the new situation. The case may be either reused to generate solutions directly or, if necessary, adapted to the changed circumstances of the current problem. The typical solution of a CBR problem would involve the following steps[43]:

1. Record the details of the current problem.

2. Match these details against the details of stored cases to find similar problem situations.
3. Select the stored solution to the current problem.
4. Adapt the stored solution to the current problem.
5. Validate any new solution and store the details of the new case.

These problem-solving steps give rise to the problem solving cycle of a typical CBR system as illustrated in Fig. 2.6.

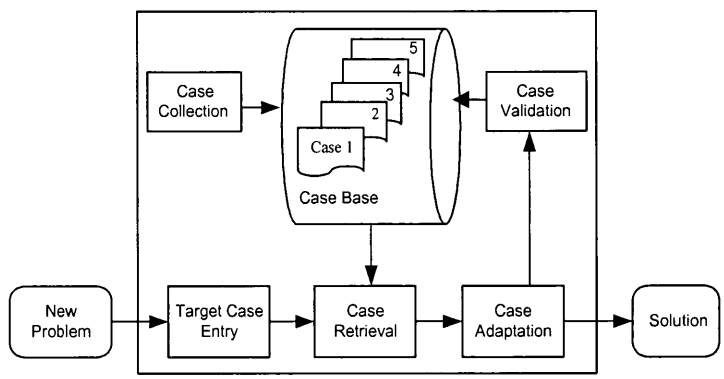


Figure 2.6: *The CBR problem solving cycle*

A new CBR problem is solved by retrieving one, or more, previously stored cases, reusing the case in the problem under consideration, revising the solution based on this reused case, and retaining the new case by incorporating it into the stored knowledge base of cases. Once a matching case is retrieved, a CBR system will attempt to reuse the solution suggested by the retrieved case.

In some circumstances, the retrieved solution case will not be sufficient. In such cases, the CBR system must adapt the solution stored in the retrieved case to the needs of the current case. Several theoretical methods have been used in CBR for adaptation[44], however detailed consideration of the theory of CBR is beyond the scope of the thesis, which is concerned with the selection, and application of an appropriate CBR strategy.

Many commercial CBR development tools are now available[45]. These include Case Advisor, Casepower, CBR Express, Casepoint, Eclipse, Caspian, and CBRWorks. Many CBR tools are capable of running on PCs. Some CBR tools are application specific and have their own user interface and therefore can be expensive.

Advantages of CBR over traditional expert system:

As already seen, the rule-based expert systems attempt to solve problems by using a combination of rules in an appropriate chaining mechanism. This approach relies on having access to the problem solving knowledge together with knowledge of the application domain[46]. On the other hand, the CBR approach does not require detailed knowledge of the domain, since CBR only needs to identify whether a similar problem has been solved previously. This gives the following advantages in applying CBR[47]:

- Reduction in knowledge acquisition, since no explicit model of the workings of the domain is necessary.
- Enhanced maintenance capability because changes to domain knowledge can be updated with new solved cases.
- Learning by acquiring new cases, without having to add new rules or modify existing rules.

CBR applications:

Case based reasoning first appeared in commercial tools in the early 1990's and since then has been sued to create numerous applications in a wide range of domains[48]:

- Diagnosis: case-based diagnosis systems try to retrieve past cases whose symptom lists are similar in nature to that of the new case and suggest diagnoses based on the best matching retrieved cases. The majority of installed systems are of this type and there are many medical CBR diagnostic systems.
- Help Desk: case-based diagnostic systems are used in the customer service area dealing with handling problems with a product or service.
- Assessment: case-based systems are used to determine values for variables by comparing it to the known value of something similar. Assessment tasks are quite common in the finance and marketing domains.
- Decision support: in decision making, when faced with a complex problem, people often look for analogous problems for possible solutions. CBR systems have been developed to support in this problem retrieval process (often at the level of document retrieval) to find relevant similar problems. CBR is particularly good at querying structured, modular and non-homogeneous documents.
- Design systems: to support human designers in architectural and industrial design have been developed. These systems assist the user in only one part of the design process, that of retrieving past cases, and would need to be combined with other forms of reasoning to support the full design process.

In the following Chapters, the use of CBR will be highlighted within the whole system design and implementation process of a particular electronic system.

2.4.4 Expert Systems in Electronics Design

Expert systems have been used in the electronic design domain to help users in making a choice of different design categories as well as for applications such as analysis, modelling, design, testing, diagnosis and control[49,50,51,52,53,54].

In the last two decades, expert systems have been widely applied for the analogue and digital electronic design applications such as filters, amplifiers, power converters and so forth.[55,56,57]

Case based reasoning[58,59,60,61] has been successfully applied in the engineering domains, among these are:

READEE: developed by Oehler and others[62], this is an interdisciplinary project carried out by a consortium of experts in electrical engineering and circuit design as well as information scientists, the application domain of the prototype being Digital Signal Processors (DSP). This first version of the READEE system can be seen as a selection tool for circuit designers that guides an engineer in selecting a DSP suitable for their application using similarity-based retrieval methods. The aim of this prototype is to prove the applicability and usefulness of the problem solving methods that have been developed in the context of the project.

CADET[63]: a case based reasoning system that functions as a designer's assistant for mechanical design. It retrieves and reuses previous successful designs while avoiding previous failures such as poor materials or high cost.

CADSYN[64]: developing case based reasoning for structural design. The CADSYN project's objective was to develop a design system that adapts a recalled case automatically to fit a new context. That is, CBR could provide a process model for generating new designs, not just recalling relevant ones.

Other recent research in using an artificial intelligence approach in electronic design was authored by John R. Koza et al.[65] who used genetic programming in an evolutionary process of electronic design. The process involves copying, crossover and mutation. The crossover operation takes two circuits and swaps some of their components, producing two new circuits. Mutation in contrast, would introduce a small change to the circuit.

Given developments in electronics, computer tools and the artificial intelligence (AI), it may be expected that expert systems will have an increasing role in the design process in coming years. As modern circuits design get more and more complex and difficult to handle by

designers, the use of expert systems based on deign reuse will be of great benefit in terms of design cost and development time.

---

[1] Bradley, D. and Dorey, A. P. (1994). Measurement science and technology- essential fundamentals of mechatronics (review article), *Measurement Science and Technology,* (5): pp.1415-1428

[2] www.intel.com. (accessed Feb. 2003). A history of the microprocessor.

[3] www.icknowledge.com, (accessed Feb. 2003). History of the integrated circuit.

[4] Herbst, L. J. (1996). *Integrated Circuit Engineering.* Oxford University Press.

[5] Chen, W. K. (2000). *The VLSI Handbook.* USA: CRC Press LCC.

[6] ibid., P12

[7] ibid.

[8] ibid.

[9] ibid.

[10] Shepherd, P. (1996). *Integrated circuit design fabrication and test.* 1st edition. UK: Macmillan Press Ltd.

[11] Smith, Michael J. S. (1997). *Application Specific Integrated Circuit.* USA: Addison-Wesley.

[12] Multisim design tool.

[13] Storey, N. (1998). *Electronics a system approach.* 2nd edition. UK: Addison Wesley Longman Ltd.

[14] Whitaker, J.C. ed. (2002). *Electronic system maintenance handbook.* 2nd edition. USA: CRC Press LLC.

[15] op. cit. 11, P345

[16] ibid., P346

[17] ibid., P379

[18] ibid., P479

[19] Awad, E. M. (1996). *Building expert systems: principles, procedures, and applications.* USA: West Publishing Company.

[20] Darlington, K. (2000). *The Essence of Expert Systems.* UK: Pearson Education Limited.

[21] Durkin, John. (1994). *Expert Systems Design and Development.*USA: Macmillan.

[22] Giarratano J. and Riley, G. (1998). *Expert Systems Principles and Programming.* USA: PWS Publishing Company.

[23] Wang, S. J, Lee, Y. S. and Siu, K. W. (1997). Expert System Aided Design, Simulation and optimization of Power Converter. *23rd International Conference on Industrial Electronics, Control and Instrumentation*, pp1016-1021, IEEE, NewYork, USA

[24] Samardar, M. (1991). EXACT: EXpert in Analog Circuit Topology. *Expert Systems with Applications,* Vol. 2, pp. 137-152, Pergamon press Plc.

[25] op. cit., 19

[26] ibid.

[27] op. cit., 20

[28] Bradley D., et al. (2000). *Mechatronics and the design of intelligent machines and systems*. UK: Stanley Thornes.

[29] op. cit.,19

[30] ibid.

[31] ibid

[32] Matlab signal processing toolbox. (2002). The Mathworks, Inc.

[33] op. cit.,19

[34] op. cit.,20

[35] Beckman, T. J. (1991). Selecting Expert System Applications. *AI Expert*, Feb., pp. 42-48.

[36] Smith, P. (1996). *An introduction to knowledge engineering*. International Thomson Computer P.

[37] op. cit.,19

[38] Schank, R and Slade, S. (1991). The Future of Artificial Intelligence: Learning From Experience. *Applied Artificial Intelligence.* 5(1) 97-107.

[39] Kolodner J, Simpson R and Sycara K. (1985). A Process Model of Case based Reasoning in Problem Solving. *Proc. IJCAI-85,* PP284-290. Morgan Kaufmann.

[40] Schank R (1982). *Dynamic Memory: A Theory of Learning in Computers and People.* Cambridge, UK: Cambridge University press.

[41] Kolodner J (1992). An Introduction to Case Based Reasoning. *Artificial Intelligence Review.* 6, 3-34.

[42] Riesbeck C and Schank R (1989). *Inside Case-Based Reasoning.* Lawrence Erlbaum Associates, Inc., Publishers.

[43] op. cit.,20

[44] Watson, Ian. (1997). *Applying case based reasoning: techniques for enterprise systems.* USA: Morgan Kaufmann Publishers, Inc.

[45] Darlington, K. (2000). *The Essence of Expert Systems*. UK: Pearson Education Limited.

[46] ibid.

[47] op. cit.,44

[48] Artificial Intelligence Applications Institute, School of Informatics - The University of Edinburgh. *http://www.aiai.ed.ac.uk*, (accessed August 2004).

[49] Sheu, B. J., Fung, A. H. and Lai, Y. N. (1988). A knowledge-based approach to analog IC design. *IEEE-Transactions-on-Circuits-and-Systems*. Feb. ; 35(2): 256-8

[50] El-Turkey, F. and Perry, E. (1989). BLADES: An Artificial Intelligence Approach to Analog Circuit Design. *IEEE Transaction on Computer aided Design,* Vol. 8, No. 6.

[51] Wu, J. G, et al. (1990). A model-based expert system for digital system design. *IEEE Design& Test of Computers,* vol. 7, no. 6, pp. 24-41

[52] Samardar, M. (1991). EXACT: EXpert in Analog Circuit Topology. *Expert Systems with Applications,* Vol. 2, pp. 137-152, Pergamon press Plc.

[53] Kokozinski, R., Hosticka, B. J. and Klinke, R. (1994). Rule-based adaptive configuration selection for analog design systems. *Analog-Integrated-Circuits-and-Signal-Processing*. March ; 5(2): pp.111-19, Kluwer Academic Publishers, Boston.

[54] Wang, S. J., Siu, K. W. and Lee, Y. S. (1997). Applying expert system and fuzzy logic to power converter. *IEEE International Symposium on Circuits and Systems*. Hong Kong, pp1732-1735.

[55] Margo, V. and Etter, D. M. (1991). An Expert system for Digital Filter Design. *IEEE Proceedings of the 23rd Midwest Symptom on Circuits and Systems.*

[56] Wang, S. J and Lee, Y. S (1996). Developing of An Expert System for Designing Analyzing and Optimizing Power Converters. *19th Convention of Electrical and Electronic Engineer,* Israel, pp359-362.

[57] Babu, V. R, Mazhari, B and Hasan, M. M. (1996). An Expert System Approach to analog Circuit Synthesis. *10th International Conference on VLSI Design*. pp425-428. IEEE Comput. Soc. Press, Los Alamitos, CA, USA

[58] Aklthoff, K. and Web, S. (1992). Case-based reasoning and expert system development. *Contemporary-Knowledge-Engineering-and-Cognition.-First-Joint-Workshop-Proceedings,* 1992: 146-58: Springer-Verlag, Berlin, Germany

[59] Sycara, K. et al (1992). CADET: A case-Based Synthesis Tool for Engineering Design. *Industrial Journal of expert System.* pp157-188

[60] Vollrath I.(1998). Reuse of Complex Electronic designs Requirements analysis for a CBR Application. *Advances in Case Based Reasoning. 4th European Workshop, EWCBR-98.* pp136-147

[61] Schaaf M. et al (2002). Supporting Electronic Design Reuse by Integrating Quality Criteria into CBR Based IP Selection. *6th European Conference- Berlin.* Pp628-41

[62] Oehler, P. et al. (1998). READEE - Decision Support for IP Selection using a Knowledge-Based Approach. *IP98 Europe Proceedings*. Miller Freeman.

[63] op. cit., 59

[64] M.L. Maher and D.M. Zhang (1993), "CADsyn: A Case-Based Design Process Model,"*Artificial Intelligence for Engineering Design, Analysis and Manufacturing,* Vol. 7,No. 2, pp. 97-110.

[65] koza J. R. et al. (2003). Evolving Inventions. *Scientific American, Inc.*

Linear Technology website, *http://www.linear-tech.com,* (accessed April 2001). Filters Home Page.

Chapter 3

An Expert System to Support Filter Design

3.1 Introduction

As set out in Chapter 1, the principal aim of the research was to investigate the development and implementation of an expert system to be used to provide assistance in designing the electronic component of a mechatronic system. That electronic system was itself the result of the decomposition of a larger system within the context of the overall electronic system design environment (ESDE) proposed previously[1,2]. In this Chapter, the structure of an expert system to support filter design will be investigated, the knowledge for which is to be acquired from human experts and other relevant knowledge sources such as texts and articles[3,4,5]. This knowledge is then coded into a form such that the expert system can retrieve the required data to support the design process. The associated filter design database is structured to enable different filter designs to be represented through a unified filter design template.

Once the knowledge base is established, the inference component of the expert system is developed, encompassing the techniques by which the expert system acts to solve the problem. A combination of case based reasoning[6] with a rule-based structure was eventually adopted after investigation as being the most suitable approach, enabling design reuse and design adaptation. The final part of the Chapter will deal with crisp and fuzzy sets and how these may be used in setting the optimising criteria.

3.2 Filter Design as An Exemplar

As indicated in Chapter 1, the electronic part of a mechatronic system represents the generic problem domain of this project, but in order to proceed the design of such a large complex system, a further breakdown of the system is needed. A top-down approach has been used to break the generic system down into its smaller implementable submodules. Most of the implementable electronic submodules fall into one or more of the categories: amplifier, microprocessor, microcontroller, filter, digital signal processor.

The overall concept of the project can be imagined as a large expert system consisting of smaller expert systems in the form of implementable submodules. The secondary objective

of the project will be to establish how these smaller expert systems communicate with each other under the umbrella of the overall system. This will be discussed in detail in Chapter 6.

Because of the similarities in designing and implementing any expert system, the objective was to work with one of these smaller expert systems representing one of the electronic submodules. In choosing the submodule to be implemented it was first necessary to look at the advantages, disadvantages, characteristics, applications and diversity of each of the submodules necessary.

Microprocessors, microcontrollers or digital signal processors, are concerned with the software implementation of a digital system. Amplifiers can be found in most electronic system applications and occur in many forms over a large range of frequencies. They are verastile,cheap to buy and can be implemented in hardware using discrete components or in integrated circuit form. On the other hand classic amplifiers can only be used for analogue signals.

As already indicated, the option that was chosen was that of filters. In signal processing, the function of a filter is to remove unwanted parts of a signal, such as random noise, or to extract useful parts of a signal, such as the components lying within a certain frequency range. If the signal needs to be filtered, it is possible to use an analogue filter before digitisation, or a digital filter after. Thus filters are available in both analogue and digital form.

The other advantage of using filters as the development domain is that they are generally well defined, which is useful in building a working expert system free from design complexity. A further advantage of using filters is the diversity of applications in which they can be used. Filters are also available in different technologies such as passive, active and digital with all these technologies found in different filter types from lowpass, highpass, bandpass and bandstop. The differences among these types are dependent on the relationship between the pass and stopbands.

Lowpass filters are by far the most common type, and are widely used in anti-aliasing and other aspects of data acquisition and signal conversion. For a lowpass filter, the passband extends from DC (0 Hz) to f_c, as shown in Fig. 3.1 for an ideal filter.

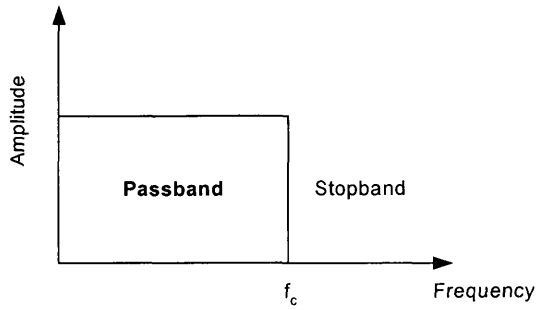


Figure 3.1: *Ideal lowpass filter*

Real or practical filters are however far from ideal. Constructed from actual electronic components, they subject input signals to operations that only approximate ideal behaviour. They provide attenuation in the stopband, but they also subject signal components in the passband to attenuation, ripple, phase shift, or delay. In addition, practical filters contain a transition region between the passband and the stopband as suggested by the shaded area in Fig. 3.2.

The roll-off rate usually expressed as the amount of attenuation in dB for a given ratio of frequencies. The ultimate roll-off rate will be 20dB/decade for every filter pole in the case of a lowpass or highpass filter.

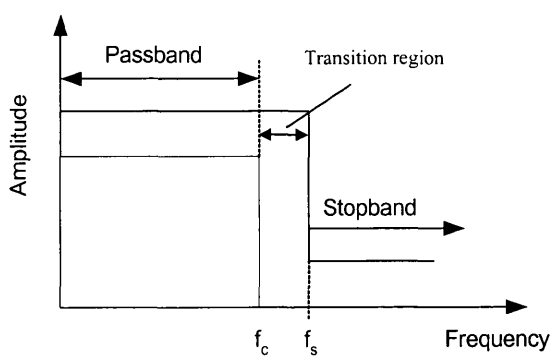


Figure 3.2: *Practical lowpass filter*

Where f_c is the cutoff frequency or passband limit.

f_s is the frequency at which the stopband begins

Filter order is directly related to the number of components in the filter, and therefore to its cost, its physical size and the complexity of the design task. Therefore higher order filters are more expensive, take up more space, and are more difficult to design. The primary advantage of a higher order filter is that it will have a steeper roll-off than a similar, lower order filter.

The "classic" filter functions were developed by mathematicians (most bear their inventors' names), and each was designed to optimise some filter property[7,8]. The most widely used are Butterworth, Chebyshev1 (Chebyshev), Chebyshev2 (inverse Chebyshev) and Elliptic. No attempt is made here to show the mathematical derivations of these functions, as they are covered in detail in numerous texts on filter theory and Appendix B sets out the formulas for different types of filter approximations. Figure 3.3 shows the amplitude response curves of a lowpass filter for different filter types of the same order.

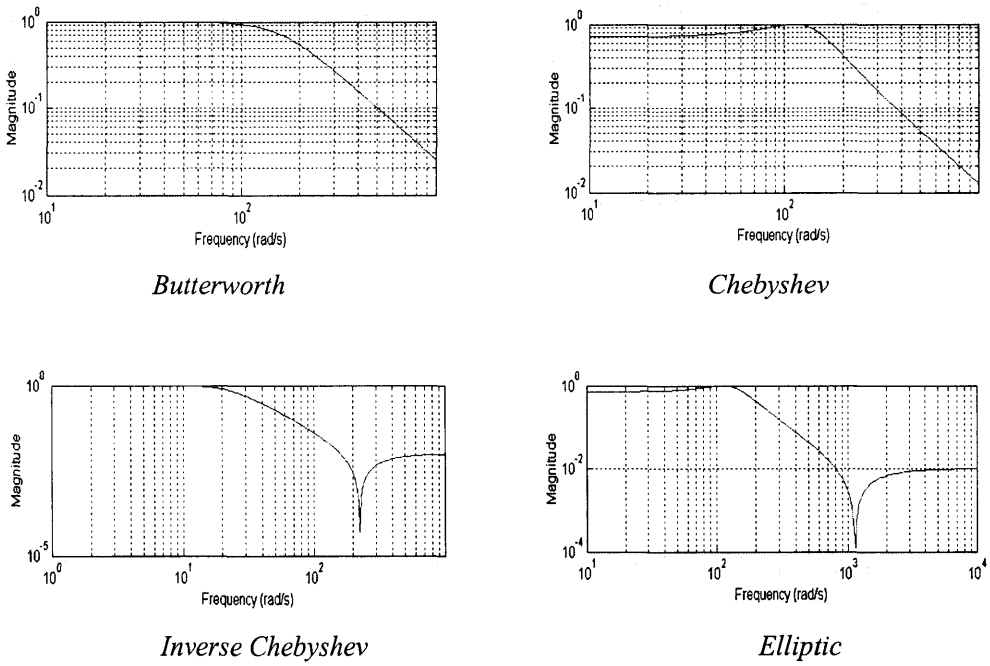


Figure 3.3: *Amplitude response curves for various second order filters with cutoff frequency of 1000rad/s*[9]

3.3 Knowledge Acquisition for Filter Design

The objective of knowledge acquisition is to compile a body of knowledge on the problem of interest such that it can be then encoded into the expert system. The most dominant source of knowledge is generally the human expert. Several acquisition tools are available to help the knowledge engineer in their task with each tool being used for a specific purpose. The most common way is to interview the domain expert.

The role of the experts at the early stage is not only to provide knowledge about filter design, but also to provide information about the ways in which the problem can be tackled. The input received from the experts will be used in the understanding of how the knowledge will be handled and also how it can be fitted into the system in order to achieve the overall system requirements.

This section begins by discussing the preparation of interview questions in the form of filter design scenarios used with human experts in order to get feedback. It then considers the other sources of knowledge used as a consequence of the results obtained from interviewing the human experts. The work with the human experts also served as a feasibility study for the project as their responses would be evaluated not just in the context of their ability to provide the necessary expertise, but also as an indicator as to the viability of the approach

Design Scenarios:

To ease knowledge acquisition in the project, filter design scenarios were set for differing filter applications. These scenarios were generated to enable a wide feedback from the experts and involved application, frequency range, volume and cost. The expert answers should involve the filter design choice, implementation, complexity, response type and power consumption among others. Initially about eight scenarios were set, two different filter design applications for each filter type.

The design requirement in each scenario was deliberately kept general in order to get conditional answers or many answers for one question, which would then serve to indicate the general approach used by a designer. The filter design scenarios are set out in Table 3.1.

Table 3.1: *Filter design scenarios*

Anti-aliasing filter	An analogue voice-frequency signal used in telephone communication system, which occupies a band from 300-3400Hz, is to be transmitted over a binary PCM system. The design is to be minimum cost for medium volume production.
Lowpass filter	A filter to suppress noise and unwanted signals for an audio signal amplifier working in a bandwidth of 10 kHz. The design is to be minimum cost for medium volume production.
Flicker	A filter is required to reject low frequency and flicker noise occurring below 1 kHz in a BJT amplifier system. The design is to be minimum cost for medium volume production.
Highpass Filter	A high pass filter to be used to select frequencies above 3MHz. The design is to be minimum cost for medium volume production.
Modem	A narrow band pass filter is to be used in FSK modem. The input signal has f (space)=4kHz and f (mark)=2kHz. The design is to be minimum cost for medium volume production.
Tuner	A band pass filter to be used for in input tuned circuit in a medium wave radio receiver for a frequency band of 530-1600 KHz. The design is to be minimum cost for medium volume production.
Audio	A filter to be used in audio equipment to reject 50Hz interference signal coming from an AC motor. The design is to be minimum cost for medium volume production.
Bandstop	A band stop filter used in a communication network consists of 20 channels, to stop the allocated signal frequency to each channel from going to the rest of the network. The design is to be minimum cost for medium volume production.

Six members of staff at the University of Abertay Dundee who are involved in electronic engineering were approached for the purpose of acquiring knowledge from them through an interview. Because of constraints on their time, the plan was to give each of them two design scenarios to be handed back before start of the interview. It was hoped that this would give them enough time to look at the design scenario and hand back the designs at their pace. Once the designs were received from the members of staff, they would be reviewed and analysed and any unclear points identified to be discussed with the member of staff.

Unfortunately, only two of the staff members were able to respond fully. The time from giving the members of staff the scenarios and the time arranged to meet for the interview was extended by between four to five weeks in order to meet with other commitments. The interviews lasted about an hour with each member of staff where the design scenarios were discussed and points cleared, as well as more suggestions being obtained. These were recorded to be reviewed at later stage. At the end of the interviews, it was pointed out that there might be more meetings in order to obtain more comments and suggestions and both of the members of staff agreed. After the interviews, the knowledge obtained was organised and refined with help from the relevant expert and a further interview was held on the phone in order to finalise the refined design. When the responses from the human experts were considered and analysed it was felt that an approach based on a conventional rule-based expert system might prevent a user from being part of the system interaction and the system would appear to work in isolation from the user.

Taking into account the feedback from the experts, and after considering alternatives, a case based strategy was therefore decided upon where an integration of rule-based and case based reasoning is used. Although, the rule-based system is slow and prone to error[10], it is flexible and can produce nearly optimal answers. The case based reasoning approach is quick as it is based on design reuse and case adaptation, and therefore the case base can be represented by the design cases stored in the database that are grounded in actual experience. By applying this integrated approach, the system will able to capture both forms of reasoning complementarily.

Also there were difficulties in obtaining further knowledge from the human experts due to their availability and their ability to engage in the knowledge engineering process. This is solved when using the case based reasoning approach as it allows other sources of knowledge to be used more effectively to form the design cases, while supporting the capture of expertise through the creation of new designs and setting up the system help.

One of these additional sources was filter design texts, where many design examples are illustrated and some of the filter design reference books used are listed in the reference

section. Another source of knowledge was online design sites. Companies such as Linear Technology[11] and Schematica[12] specialise in filter design and products, and filter design cases were obtained from their web sites complete with design details and supporting information, including specifications and schematics. Other similar web sites are given in the references[13].

3.4 Knowledge Representation

Once knowledge has been acquired, the next step is that of knowledge representation, coding the knowledge in such a way that an inference program will be able to process the knowledge in order to draw conclusions.

This section begins by exploring the hierarchical structure of filters. This information, along with the knowledge acquired and discussed in section 3.3, can then be used in coding the knowledge using production rules or frames. Each frame can in turn represent a design case in the database. It is explained how these frames will be extended to use a unified filter design template to capture different filter designs. This template represents a filter as an object and the attributes or fields needed by different filter designs are stored in a database file, which can be accessed by the inference program.

3.4.1 Hierarchical Structure of Filters

To ease representation of the acquired filter design knowledge into frames and rules, the concept of a filter needs to be structured in a hierarchical form. This structure organises the concept of a filter at different levels of abstraction as shown in Fig. 3.4.

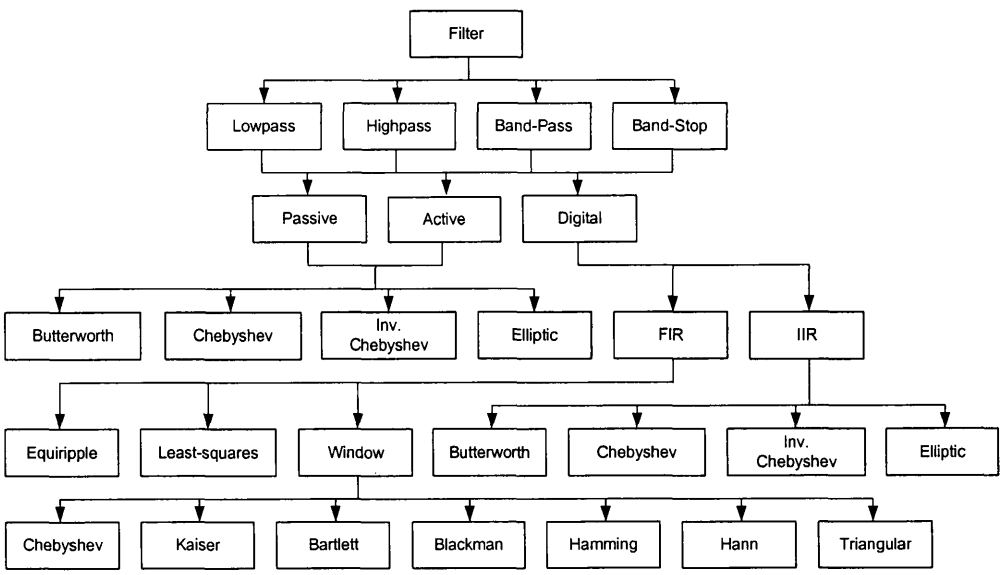


Figure 3.4: *Filter hierarchy*

The top-level contains information common to all filters. The lower levels contain information specific to their retrieval category. The bottom level is the specific filter instance. Each instance inherits information from the top-level class "filter" and also features from their associated subclasses. The representation of the acquired filter design knowledge in frames and rules will be based on this structure.

Knowledge Representation Techniques:

Knowledge captured from experts and other sources must be organised in such a fashion that a computer inferencing program will be able to access this knowledge when needed and use it to draw conclusions. As discussed, there are several methods of representing knowledge in an AI system[14,15] and production rules and frames were selected here for implementation.

Production rules:

The majority of expert systems use rules, which is why many expert systems are referred to as rule-based systems, to represent knowledge[16]. Rules may be constructed either through examples (called scenarios) or using an IF ... THEN format for knowledge capture. The example used earlier in the Chapter can be used again here:

IF "*An analogue to digital converter is to be used*"
THEN "*An anti-aliasing filter is required*"

As already indicated, this rule could then be linked with other rules such as

IF "*An anti-aliasing filter is to be used*"
THEN "*An analogue lowpass filter is required*"

The IF... THEN format rules will be constructed in Chapters 4 and 5, dealing with system design and implementation. The system will be modelled using process modelling at different levels and implemented by reference to individual process. These can in turn be represented in the process specification in an IF ... THEN format using structured English. The implementation program will consist of a structured list of IF ...THEN rules for different processes, starting from the lower levels and finishing at the top-level of the system modelling process.

The advantage of using rules is that they are easy to understand and they are communicable because they are natural form of knowledge. Modification and maintenance are relatively easy and each rule is usually independent of all others.

Frames:

Marvin Minsky[17] first conceived the concept of frame in 1975 when he defined a frame as follows:

"A frame is a structured piece of information about the properties, characteristics or features of an object, act or event".

A frame serves as a kind of template for holding related clusters of data. This data is usually stored in slots, which represent attributes of that object. For example, the frame for an object called filter might have the slots and associated values as shown in Fig.3.5.

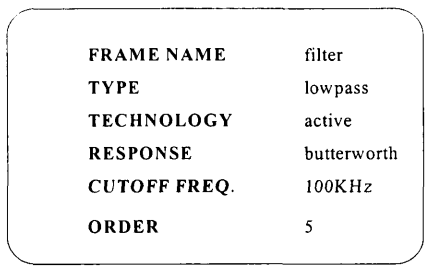


Figure 3.5: *Filter frame*

A class frame represents the general characteristics of some set of common objects. Figure 3.5 shows an example of a class frame of type filter. The other fields shown are the properties describing the general characteristic of most filters. When all the slots are filled with values, the frame is considered *"instantiated"*, which means an instance of the frame is created. When an instance of a class is created, the frame inherits both properties and property values from the class. In the example above, consider the frame of " lowpass" which is an instance of the class " filter". Since lowpass is a filter, it inherits all the properties of filter in Fig. 3.5.

Referring to the hierarchical structure of filters of Fig. 3.4, the top-level frame contains information common to all filters. The next lower level frames (called subclasses) contain information more specific to their individual category or type of filter. For example, lowpass and highpass filters share features common to all filters and each has some unique features that serve to discriminate between them, such as their frequency range. The lower level frames are the specific filter instances. Each instance inherits information from the top-level "filter" frame, and also features from their associated subclass. For example, a filter instance "Hamming " in Fig. 3.4 inherits information from the top-level " filter" and features from the subclasses of specific filter type, digital filter, FIR and window. In general, an instance inherits information from its parent, grandparent, etc.

The filter design knowledge acquired from interviewing the experts and from other sources needed to be encoded. It has been found that the assembled knowledge can be fitted into frames to which other filter attributes such as cost, implementation, component count, schematic and layout can be added, and each frame will therefore represent a design case in the database.

3.4.2 Database of Filter Design Cases

Filter design knowledge for each design is represented in a form or frame. On increasing the number of design cases, there is a need to document these cases in a database. The system is then able to retrieve data from the database as well as store new data. The database management system Microsoft Access is used here for processing the filter design data. Designs are stored in a single database table with each filter design represented as a record, which has all the fields of the design.

Initially, the database was established with twelve filter design cases, sufficient to test the system. The database file can however handle hundreds of filter design records, and the higher number of existing records, the bigger the chance of obtaining a design match.

The design of the general database file as implemented comprises a single table containing the individual filter design records. Because a single table is used, the database structure is kept simple, both in structure and in the ability to establish a link to the Visual Basic User front end. The table of filter design cases is then as shown in Fig. 3.6.

Filter_cases : Table

Designer	Type	Response	Cutoff_Freq	lower_cut	upper_cut	passband	stopband	Order	Application
Schematica Software	Lowpass	Elliptic	50000			0.1	10	7	Antialiasing
Schematica Software	Highpass	Elliptic	400			0.9	33	4	1/f noise elimination
Schematica Software	Bandpass	Elliptic		50000	60000	0.05	25	6	frequency tuning
Linear technology	Lowpass	Butterworth	11				10	4	Antialiasing
Linear technology	Lowpass	Elliptic	20000			0.1	10	4	phased locked loop
Dailey	Highpass	Butterworth	500				40	2	1/f noise elimination
C.Main	Highpass	Chebyshev2	1000				40	5	dc blocking
Linear technology	Highpass	Chebyshev1	30000			0.05	30	8	noise elimination at low frequenc
linear Technology	Highpass	Elliptic	11			0.1	23	8	highpass
S.Reynolds	Bandpass	Butterworth		2000	2100		40	4	graphic equiliser
Linear technology	Bandpass	Elliptic		96000	104000	0.2	25	8	frequency tuning
C.Main	Bandstop	Butterworth		47.5	52.5		44	2	reject 50Hz power line noise

(a) *Table part (I)*

Filter_cases : Table

Implementatio	Volume	Component Co	Complexity	Chip features	Operating pow	Cost	Opertating Tem	Notes	Reference
Discrete compor	high	30	high	general usage	+/-3.5 to +/-18V	medium	0 - 70 degree	Cascaded of 2nd a	Schematica Softwar
Discrete compor	high	19	low	general usage	+/-3.5 to +/-18V	low	0 - 70 degree	Cascaded two 2nd	Schematica Softwar
Discrete compor	medium	28	medium	general usage	+/-3.5 to +/-18V	medium	0 - 70 degree	Cascaded of 2nd o	Schematica Softwar
Discrete compor	low	9	low	extremely easy	3 to +/-5V	low	-40 to 85 degree	Cascadable to form	www.linear-tech.com
Discrete compor	medium	16	medium	very low noise	-5V to +5V	low	-40 to 85 degree	low distortion	www.linear-tech.com
Discrete compor	low	5	low	precision	+/-3.5V to +/-18\	low	0-70 degree	Sallen-Key	text book: operation
discrete compor	high	12	low	general usage	+/-3.5V to +/-18\	low	0-70 degree	higher order can be	Dr. C.Main
discrete compor	medium	15	high	very low noise	-5V to +5V	high	-40 to 85 degree	low distortion	www.linear-tech.com
discrete compor	low	18	high	very low noise	-5V to +5V	high	-40 to 85 degree	low distortion	www.linear-tech.com
discrete compor	medium	14	low	general usage	+/-3.5 to +/-18V	low	0 - 70 degree	obtained by cascac	text book: operation
discrete compor	low	19	high	very low noise	-5V to +5V	high	-40 to 85 degree	low distortion	www.linear-tech.com
discrete compor	low	9	low	precision	+/-3.5V to +/-18\	low	0-70 degree	single null used for	text book: electronic

(b) *Table part (II)*

Figure 3.6: *The database table for filter design*

Because of the size of the table with 20 fields per record, Fig. 3.6 is divided into two parts in order to fit the page at a suitable level of detail. Although some of filter background was discussed in this Chapter, the meaning of the individual fields of Fig. 3.6 is as set out in Table 3.2:

Table 3.2: *Fields description*

Field	Type	Description
Designer	Text	Gives the source of the design. This may be either a human expert, the author of filter design book, a filter design company and so on.
Type	Text	Defines filter type, may be lowpass, highpass, bandpass or bandstop.
Response	Text	Encompasses 4 responses; Butterworth, Chebyshev1, Chebyshev2 and Elliptic.
Cutoff frequency	Number	To indicate the passband limit for LP and HP filters.
Lower cutoff	Number	The lower corner frequency in the passband.
Upper cutoff	Number	The upper corner frequency in the passband.
Passband ripple	Number	When the filter is not monotonic within its passband.
Stopband attenuation	Number	Minimum allowable attenuation within the stopband.
Order	Number	In the range 2-8, higher order filters can be formed by cascading lower order filters[18,19]
Application	Text	Filter applications include; antialiasing, phase locked loop, noise elimination, frequency range selection, dc blocking and so forth.
Implementation	Text	Contains information on the physical circuitry, which here is limited to discrete components; common for active filters that consist of resistors, capacitors and integrated circuits.
Volume	Text	Provides the production volume of the filter design.
Component count	Number	The number of components in each filter.
Complexity	Text	Related to filter order and component count to set the design complexity as low, medium or high. This is essentially a function costing approach as set out by French[20].
Chip features	Text	For information purposes.
Operating power	Text	For information purposes.
Cost	Text	This provides a cost measured for the filter. See Section 6.2.1 below
Operating temperature	Text	For information purposes.
Notes	Memo	Enables a short note of the characteristics of the design such as low distortion, low cost or cascadable to be entered.
Reference	Memo	Provides the source of the design, for instance if it were a textbook, then the title of the book would be entered.

Microsoft Access was chosen in the thesis as a general database management system and it found to be appropriate for implementing the prototype. For developing a large database system, the same principal used in the prototype can be applied. If using Microsoft Access it may be necessary to consider avoiding some of the general features that may slow the search process or to consider the use of other databases such as Oracle which are designed to handle large volume of data.

Designing a template:

Filter design knowledge has been represented in a frame. It is possible now to extend that frame to a unified filter design template to capture different filter designs. The template consists of a filter as an object and the attributes or fields as needed by different filters.

Fields include designer name, filter type, application, cut-off frequency, implementation and others as shown in Fig.3.7.

At this stage, the fields in Fig. 3.7 are distributed according to the filter hierarchy of Fig. 3.4, which itself is based on the properties of inheritance. For instance, if the filter upper frequency is to be selected then the filter types, lowpass or highpass may inherit this. Or if the filter response 'Elliptic' is chosen, then this will inherit the passband and stopband attenuation. This is considered in Chapters 4 and 5 when the system design and implementation is covered.

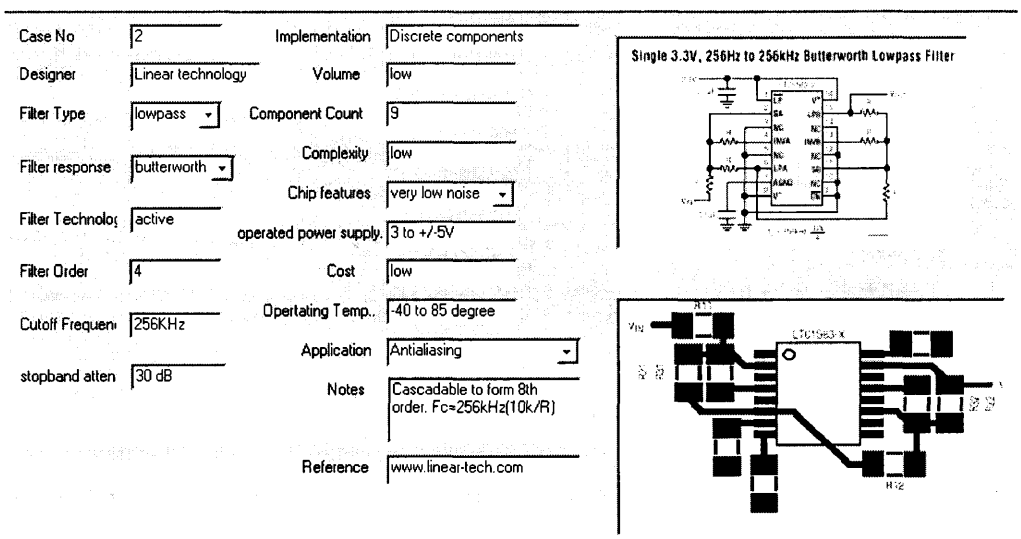


Figure 3.7: *Filter design template*

Some fields in the demonstration database need fixed values to avoid system complexity. For instance, the implementation field has been chosen as discrete component for all records. Fields such as volume, complexity and cost are to be used for optimisation and two options were considered for their evaluation; conventional crisp sets and fuzzy sets. These alternatives are discussed in detail in Section 3.5.

Testing the database file:

The database file represents the transfer of the work on knowledge acquisition to knowledge representation. As the filter design cases stored in the database file will be used later on by the system to retrieve a particular design, it was important to test the database file at an early stage to identify and remedy problems before integrating it into the overall system. The objective of the test is to check that each filter design case in the database has complete design fields to assure that the design is consistent, and ready to be searched by the inference engine.

Validation of the database was done by checking each of the design cases and making sure that they matched the acquired filter design knowledge. Matlab was chosen for the purpose of simulation as it has a filter design library and can readily be linked to the database. The frequency response and phase angle diagrams (see Fig. 3.8) are checked visually against the filter design entries stored in the database file, and these in turn can be checked by reference to the filter design scenarios and the expert response.

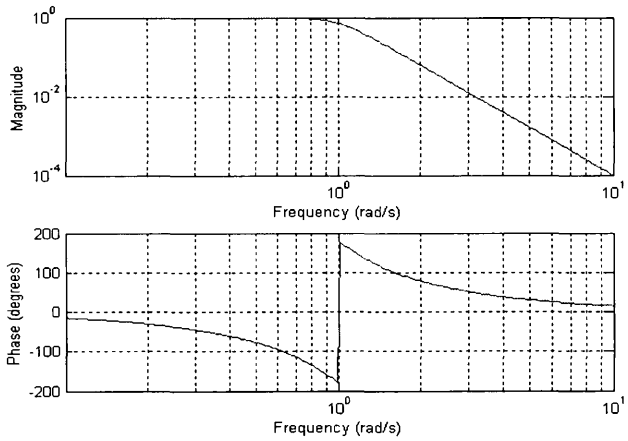


Figure 3.8: *Frequency and phase response of Butterworth low pass filter* [21]

Based on the knowledge acquired from the experts and as represented in the database file, the simplified diagram of Fig. 3.9 can be derived which represents the relationship of the expert system to the filter design process. This simple diagram also forms the basis for the more detailed diagrams that be developed in Chapter 4 where system modelling and design are discussed.

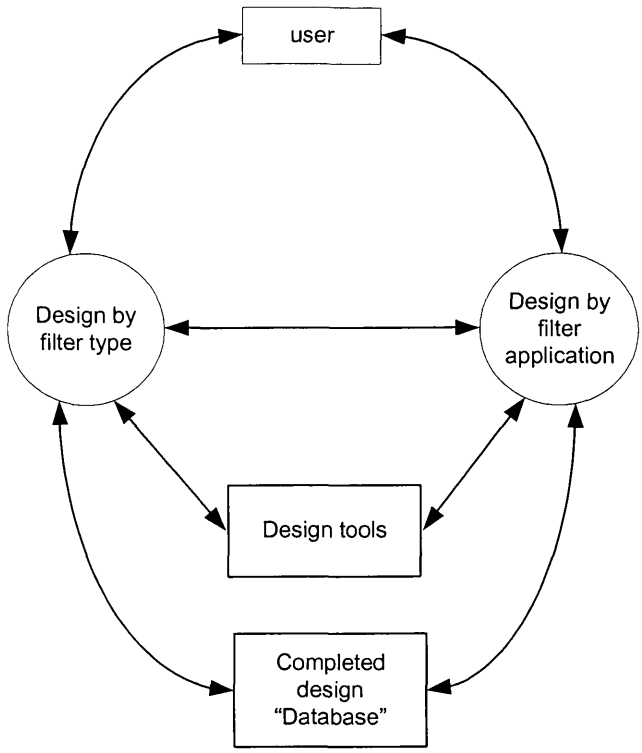


Figure 3.9: *A simplified diagram of the relationships for the filter design expert system*

3.5 Inference

Once completed, the knowledge base is ready for use. To use the knowledge base there is a need for a computer program that will access the knowledge for the purpose of making inferences and decisions and for problem solving. This program is effectively an algorithm that controls elements of the reasoning process and it is usually referred to as the inference engine or control program[22]. The most popular strategies are based on some combination of forward and backward chaining, as discussed in Chapter 2. The inference program adopted here is based on a combination of rule-based and case based reasoning. The design and implementation of the system and the related used approaches are presented in detail in Chapters 4 and 5.

3.5.1 Crisp or Fuzzy Sets for Optimisation

In this section, approaches to setting the optimising criteria for a filter design such as 'Cost', 'Component count', ' Volume' and 'Complexity' based on crisp and fuzzy sets, and the relationships between them are discussed and evaluated.

Crisp Sets:

Using a crisp sets, it may chosen to define a design cost of less than £2 as 'Low', a design cost in the range £2 to £6 as 'Medium' and a design cost greater than £6 as 'High'. These definitions can be defined graphically as in Fig. 3.10 from which it is seen that there is no overlap between sets.

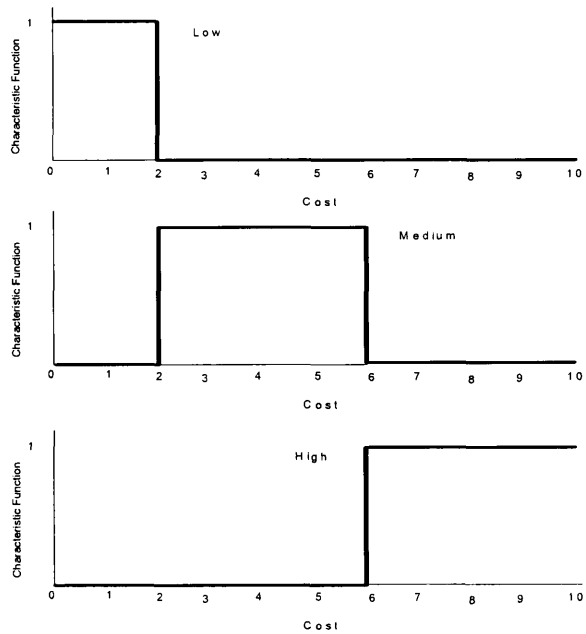


Figure 3.10: *Crisp sets for Cost*

Crisp sets imply exact knowledge, although human knowledge is often inexact. Therefore when apply crisp sets in reasoning; they only express the truth or falsity of statements, which limits the expressive power characteristic of a real world situation. It also difficult to introduce new knowledge about real life situations: only what is derived from the existing information in the knowledge base can be used in reasoning within complete certainty.

Fuzzy Logic:

A definition of fuzzy logic is provided by Durkin[23] who states that:

> *"Fuzzy logic is a branch of logic that uses degrees of membership in sets rather than a strict true/false membership."*

If instead of crisp values, in a fuzzy representation the distributions of low, medium and high cost blend into each other as illustrated by the relationships of Fig. 3.11. From this it is seen that there is a significant overlap between the sets defining 'Low' and 'Medium' and between those for 'Medium' and 'High'.

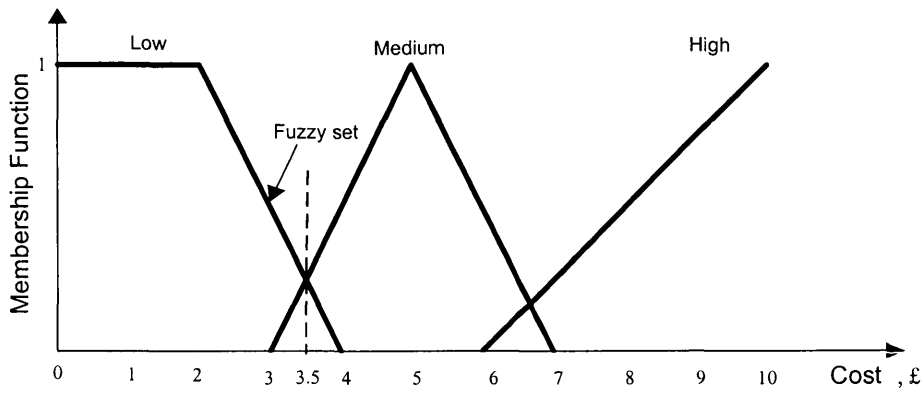


Figure 3.11: *Fuzzy sets for Cost*

Thus if the input cost variable is £3.5 then, referring to Fig. 3.11, that value is a member of the fuzzy set "low cost" with a membership value of 0.25, and at the same time a member of the fuzzy set "medium cost", also with a value of 0.25. Compared to a crisp value, a single object may thus be considered as a partial member of multiple fuzzy sets.

In order to use fuzzy logic in the search processes of the expert system, once the fuzzy variables and the fuzzy sets are defined, there is a need to define the associated fuzzy rules. A fuzzy rule infers information about the fuzzy variable contained in its conclusion from information about another variable, or variables, contained in its premise. Once the rules are inferred it is necessary to take the induced fuzzy set and obtain a crisp value. This process is called defuzzification.

When the fuzzy sets and rules are completed, the next task is to build the fuzzy system. This involves the coding of the fuzzy set together with the rules and procedures for performing the associated fuzzy logic functions. To accomplish this, there are two main routes; one is to build the system from scratch using an appropriate programming language and the second is to use a fuzzy logic development shell. Given its availability and to save time, the second option was selected using the Matlab Fuzzy Logic toolbox[24]. This provides a quick way of evaluating the process of using fuzzy logic for the system design variables of cost, volume and complexity.

Once the variables are named, and the membership functions given appropriate shapes and names as shown in Fig 3.12 for the input variable 'cost'. The rule editor then allows the construction of the rules together with any weights to be applied.

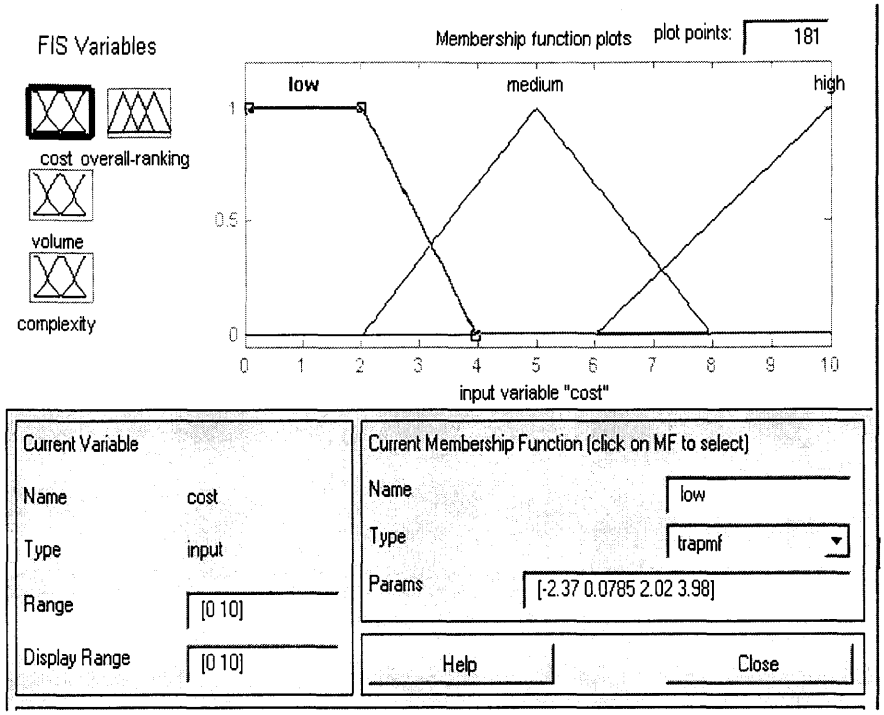


Figure 3.12: *The membership function editor*[25]

The input variables along with the associated membership functions define the number of possible rules that can be applied. In this case three input variables each with three values of membership function gives $(3)^3 = 27$ possible rules. Some of these rules are shown in the upper part of the rule editor screen shown in Fig. 3.13.

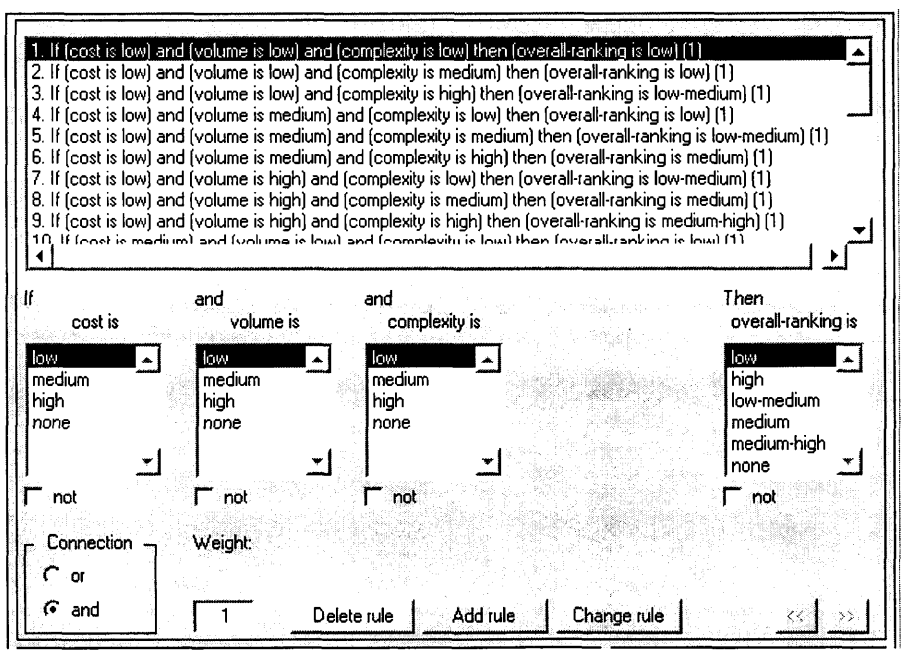


Figure 3.13: *The rule editor*[26]

The most common process of defuzzification and that, which is used here to generate the overall ranking, is the centroid calculation, which returns the centre of the areas the output function.

As an example, if the cost value is 3, which lies in both the 'low' and 'medium' membership functions, the volume is 5, which lies in the 'medium' membership function and the complexity is 7, which is represented by the 'medium' and 'high' membership functions then the overall ranking output value after the fuzzy inference process of aggregation and defuzzification will be 4.27. Therefore using fuzzy logic, the resulting output value will be more representative of the actual conditions.

Other advantages of using fuzzy logic are:

- It is conceptually easy to understand.
- It is flexible.
- It can be built on top of the experience of experts.
- It is tolerant of imprecise data.
- It can model nonlinear functions of arbitrary complexity.

On evaluating the crisp and the fuzzy logic techniques, it was concluded that a fuzzy logic implementation would be far more efficient and beneficial to the system in terms of the precision and representation of the design criteria. However, it was found that in implementing a fuzzy expert system, there was an incompatibility between the Matlab fuzzy logic toolbox and the Visual Basic system application. While building the fuzzy system from scratch using Visual Basic is possible, it would take a considerable time to achieve and validate and it was decided to implement the system in the first instance using crisp values to demonstrate the approach.

Although it was decided to use crisp sets in the thesis, the evaluation of using fuzzy sets has presented the importance and possible impact of the technique on the final version of the system. It is therefore a high priority element of any future work, and one, which might be considered a project in its own right.

3.6 Summary

This Chapter investigated the structure of an expert system to support filter design. The knowledge acquisition and knowledge representation for a filter were achieved, along with a filter design database. The expert system inference component was based on techniques

enabling design reuse and design adaptation. The main outcomes of this Chapter are therefore:

- Filter design knowledge has been acquired in part from human experts. These were contacted by generating filter design scenarios, followed by an interview or interviews.
- The analysis of the responses of the human experts identified a number of limitations with a rule-based approach and their input was then enhanced and supplemented by filter design knowledge from other sources such as filter design texts and online design sites. When the responses from the human experts and other sources were considered and analysed, after evaluation it was felt that a conventional expert system based on rules was not practical and that it was more suitable to integrate a Case Based expert system, based on design reuse and case adaptation. This will ease the search for a design while supporting the capture of expertise through the creation of new designs and will also support the provision of system help.
- The knowledge acquired was coded such that the inference program will be able to process the knowledge needed to draw conclusions. Two knowledge representation techniques have been used, production rules and frames. Filter design knowledge for each design has been represented in a frame.
- On increasing the number of the design cases, a database has been designed to document these cases and to support data retrieval as well as to store new data. This has been extended to use a unified filter design template that captures different filter designs. Filters are considered as objects and attributes or fields are filled by the filter specifications.
- The database of filter design cases was tested and validated to identify and remedy any problems at an early stage. The test involved checking that each filter design case had completed design fields to ensure that the design is consistent and ready to be searched by the inference engine. Matlab was used to check the frequency and phase response of each design.
- For setting the optimising criteria such as cost, volume and complexity and in order to search the database crisp sets will be used while the fuzzy logic approach will be highlighted in any future work.

[1] Walters, R.M., Bradley, D. A. and Dorey, A. P. (1999). A conceptual study for a computer-based tool to support electronics design in a mechatronic environment. *Microprocessors and Microcomputers,* 24: pp.51-61

[2] Walters, R. M. (1996). *The high-level design of electronic systems.* PhD thesis, Lancaster University.

[3] Smith, P. (1996). *An introduction to knowledge engineering.* International Thomson Computer P.

[4] Greenwell, M. (1988). *Knowledge engineering for expert systems,* Ellis Horwood: Wiley.

[5] Hart, A. (1989). *Knowledge acquisition for expert systems.* 2nd edition. London: Kogan Page.

[6] Watson, Ian. (1997). *Applying case based reasoning: techniques for enterprise systems.* USA: Morgan Kaufmann Publishers, Inc.

[7] William, A. and Taylor, F. (1995). *Electronic filter design handbook.* 3rd edition, New York: McGraw-Hill.

[8] Temes, G. C. and Mitra, S. K. (1973). *Modern filter theory and design.* Wiley

[9] Matlab signal processing toolbox. (2002). The Mathworks, Inc.

[10] Riesbeck C and Schank R (1989). *Inside Case-Based Reasoning.* Lawrence Erlbaum Associates, Inc., Publishers.

[11] Linear Technology website, *http://www.linear-tech.com,* (accessed April 2001). Filters Home Page.

[12] Schematica Software website, *http://www.schematica.com,* (accessed April 2001). Filter Wiz active filter designer.

[13] Analog Devices website, http://www.analog.com/, (accessed May 2001). Semiconductor company specializing in high-performance analogue, mixed-signal and DSP.

[14] Durkin, John. (1994). *Expert Systems Design and Development.* USA: Macmillan.

[15] Turban, E. (1992). *Expert Systems and Applied Artificial Intelligence.* USA: Macmillan Publishing Company.

[16] op. cit., 13

[17] Winston, P. ed.(1975). *In the psychology of human vision.* McGraw-Hill, pp.211-217.

[18] William, A & Taylor, F (1995). *Electronic filter design handbook.* 3rd edition, New York, McGraw-Hill.

[19] Temes & Mitra, (1973). *Modern filter theory and design.* Wiley

[20] French, Michael J., (1990). *Function Costing: A potential Aid to Designer.* Journal of Engineering Design, Vol. 1, No. 1

[21] op. cit., 9

[22] Darlington, K. (2000). *The Essence of Expert Systems.* UK: Pearson Education Limited.

[23] op. cit., 13, p364

[24] Matlab fuzzy logic toolbox. (2002). The Mathworks, Inc.

[25] ibid.

[26] ibid.

[27] ibid.

Chapter 4

System Modelling and Design

4.1 Introduction

In the last Chapter, the knowledge acquisition and representation strategy for the filter design expert system was discussed and it is now necessary to consider the way in which the decisions reached are implemented. This Chapter therefore considers the modelling of the system using formal methods as part of the design and implementation process to establish the software and related structures.

The first part of this Chapter considers design methodology, in particular process-oriented and object-oriented methods and assesses how they can be applied to model and design the proposed design support expert system. The second part is concerned with the detail of the system, involving the design of the user interface, system dialogues and files and databases, including all fields and records.

4.2 Software Methods for System Development

An expert system can be considered as specific application software in which general software methodologies can still be used in the analysis and design process. In this section the focus will be on exploring two paradigms in software system development. The first of these is the process-oriented method, which represents software essentially as a series of processes, or functions, that transform input data into output data. The other is the Object-Oriented (O-O) method, which represents software as a collection of independent entities or objects that offer services and communicate via messages.

Both approaches will be examined to determine which is to be used in developing the expert system. Selection will be based on an assessment of the two methodologies in relation to the task of developing the required expert system. The chosen method will then be used to model and design that system.

4.2.1 Process-Oriented Methods

Process or functional modelling involves graphically representing the functions, or processes, which capture, manipulate, store, and distribute data between a system and its environment, and between components within a system. A common form of process model is

the Data Flow Diagram (DFD) and over the years, several different tools have been developed for this[1].

There are two different sets of Data Flow Diagram symbols developed by DeMarco and Yourdon[2, 3] and by Gane and Sarson[4] respectively. Each set consists of four symbols that represent data flows, data stores, processes, and sources/ sinks (or external entities). The set of symbols used in this thesis is that of DeMarco and Yourdon and is shown in Fig 4.1.

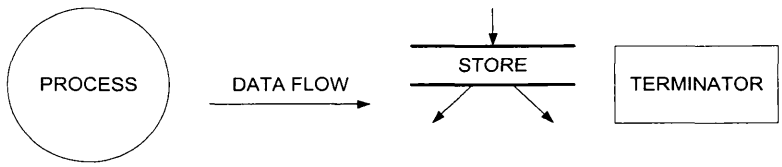


Figure 4.1: *Data Flow Diagram symbols*

To develop the system using process modelling, the following stages have to be followed:

The Statement of Purpose:

The first stage in the modelling process is to generate a brief, concise textual statement of the purpose of the system[5]. The statement of purpose for the system under consideration here is:

> *"The system interacts with the users, whether expert or novice, to set their design needs and provides help and advice to the users upon request. The system can be linked to external design tools, where the user can set the design parameters. The system can search for a design case that meets the user request. The system assists the user in accepting or modifying a design and will store it as a new design in the system database."*

The Context Diagram:

The context diagram is used to define the boundaries of the system. It shows the interactions of the system with the outside world, and the information transferred. The context diagram for the proposed electronic system is shown in Fig. 4.2, which represents the highest level (Level 1) view of the system with a single process representing the entire system[6].

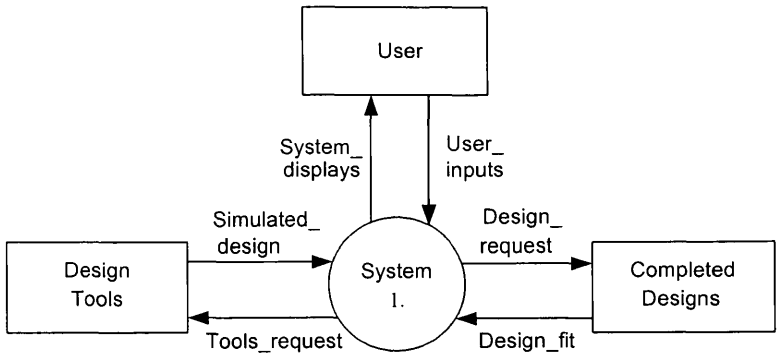


Figure 4.2: *Context diagram for an electronic design system*

Data Flow Diagram

The first stage in the development of the functional model is the production of a set of Data Flow Diagrams. These use a graphical representation to enable information about the system to be displayed clearly and concisely. The components of the Data Flow Diagram are as shown in Fig. 4.1 and defined below[7].

Process - The first component of a DFD is the process; common synonyms are 'bubble' or 'function'. A process represents a part of the system that acts to transform input data into output data. Processes are given a name and numbered using decimal notation, with each subsequent level of decomposition gaining an extra 'decimal point' in its number. Each of the lowest level processes is then included in the process specification.

Flow - A flow describes the movement of information from one part of the system to another. Each flow is given a distinct name and included in the Data Dictionary.

Store - A store models a collection of data packets at rest. Typically, the name chosen to identify the store is the plural of the name of the packets that are carried by data flows into and out of the store.

Terminator - Terminators represent external entities with which the system communicates. They are not of direct interest to the system and consequently no information is required about them other than the information they supply, and when it is supplied.

Once the diagrams have been drawn for each level, they can be linked together to provide the overall Data Flow Diagram of the system. The act of going from a top-level diagram to its lower levels is called functional decomposition[8]. This is an iterative procedure in which processes on any given diagram are explained in greater detail on another diagram representing a further level of decomposition. Decomposition continues until it reaches the point where no process or sub-process can be broken down any further. Ideally the number

of processes and associated terminators should be about six or seven on any single diagram[9].

Data Dictionary

The Data Dictionary is used to describe the composition of each of the data flows used on the diagrams. It is usually constructed using some form of structured language, an example of such an entry is:

Name	=	title +first-name + (middle-name) + last-name
Title	=	[Mr. | Miss | Mrs. | Ms. | Dr. | Prof.]
First-name	=	{legal-characters}
Middle-name	=	{legal-characters}
Last-name	=	{legal-characters}
Legal-character	=	[A-Z | a-z | 0-9 | ' | - | |]

Process Specification

The Process Specification is produced for the lowest level processes and sub-processes in the system. The method of defining the process is not as rigid as for the Data Dictionary, but is best done using some form of structured language or mathematical statements. The purpose of the process specification is to define the relationships between the inputs and outputs of each process or sub-process.

The most commonly used tool for the Process Specification is Structured English[10], which will also be used in the thesis. Structured English is a modified version of the English language that is used to specify the system processes in a Data Flow Diagram. The whole point of using Structured English to represent processes is that it is relatively easy for programmers to read and understand in order to implement the system. It is possible to use Structured English to represent all these processes typical to structured programming: sequence, conditional statements, and repetitions. Once this procedure has been completed, the implementation model can be constructed.

A full discussion of the Data Dictionary, including a list of the meanings of all the terms used and examples of Process Specification can be found in Chapters 10 and 11 of '*Modern Structured Analysis*' by Edward Yourdon[11].

4.2.2 Object-Oriented Methods

The second technique examined, which again originated as a software design methodology, is the use of object-oriented methods. An object-oriented system is viewed as a collection of interacting objects that communicate by passing messages rather than as a collection of

processes that communicate by passing data. Objects should be based on real world entities, and because of this it can be argued that object-oriented approaches have a better mapping to real world problems and hence are more easily understood[12].

The object-oriented approach to software development was developed to support the production of software that is more maintainable, testable, reusable and able to cope with large and complex systems. Object-orientation has its own specialised vocabulary, which must be understood in order to understand the methodology[13].

Objects

The basic building block of object-oriented software is the object. Software objects are derived from and model real-world objects in the application domain. These objects or things then form the subject matter of the system. Objects will have properties or attributes that are of interest to the system developer. For instance, an electronic filter object has attributes such as type, cost, frequency and response. In addition to having attributes, objects can also provide operations. An operation is a function or process that can be performed by the object[14].

There are four key concepts in the object-oriented approach: *abstraction, encapsulation, inheritance* and *polymorphism*[15]. Each of these will be considered briefly.

Abstraction - An object's name, class and the names of its operations form its public interface. This provides the abstraction, all the object's clients then need to know the information provided in the public interface, while the internal representation of the object's data and operations is irrelevant and invisible to the clients. As an illustration, the class of the filter object will have attributes such as type, technology, and cost as shown in Fig.4.3.

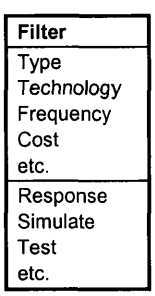


Figure 4.3: *The class for the filter object*

Encapsulation - Also referred to as information hiding, this refers to the practice of separating the external characteristics of an object, which can be accessed by other objects, from the internal implementation details of the object itself, which are hidden from other objects. There is therefore a separation between the interface to an object and its operations.

This means that objects have very low coupling and that an internal operation can be changed without having knock-on effects on the rest of the system.

Inheritance - Recognising that certain classes are special cases of other, more generalised, classes reduces the number of completely distinct cases that must be understood and analysed. This can greatly simplify the modelling of a problem and assist in its solution. In software systems it is also the basis for code re-use.

Figure 4.4 presents an object diagram, which shows that a passive filter and active filter are both a 'form of' the analogue filter class, which in turn is a 'form of' the electronic filter class. A passive filter may therefore be termed a sub-class of the analogue filter class. The analogue filter class may then be termed a 'super-class' of the passive filter class.

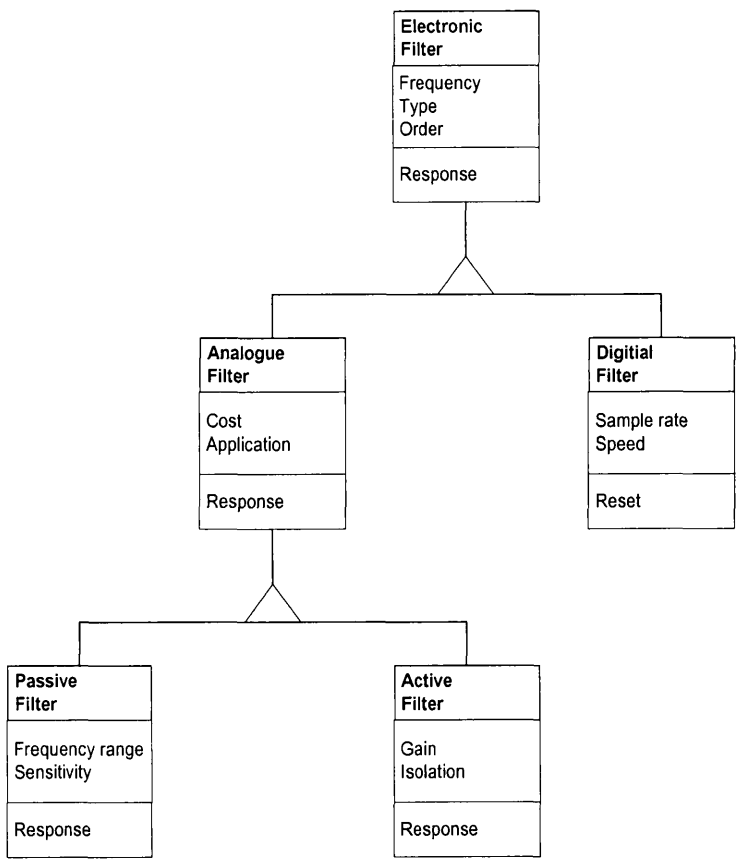


Figure 4.4: *Inheritance of attributes and operations of a filter*

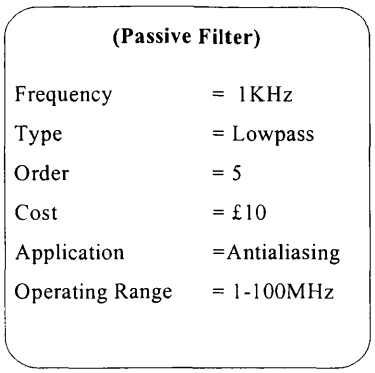


Figure 4.5: *An example of object instance*

In Fig. 4.4 note how the attributes and operations have been distributed. The electronic filter class has attributes such as 'frequency ' and 'type', which are common to all electronic filters. The analogue filter class inherits those attributes from the super-class and then adds further attributes which are specific to all analogue filters, and so on. Fig. 4.5 gives an example of a passive filter instance, which shows attributes corresponding to all its inherited attributes in addition to its own.

Polymorphism - The term polymorphism refers to the ability to define program entities such as operations that take more than one form. Polymorphism linked with an inheritance hierarchy allows a single message to be interpreted in different ways by different objects [16].

It can be seen from Fig 4.4 that because of inheritance, all electronic filters should be able to carry out operations consistent with achieving the desired response. Obviously, the most appropriate response for an analogue filter will be different from that for digital filter. The specific routine for each different class is known as a method. Thus operations are implemented by methods

Object-Oriented methodologies

There exists a wide range of object-oriented methodologies[17], each with its own view of the development process. Some focus on design and implementation, some on analysis, some concentrate on a particular modelling technique, while others are specifically geared towards a particular implementation language. However, there is one feature that all these methodologies have in common; they organise the development of systems around objects. Because of the similarities between the object-oriented methodologies, only one of the most popular methods will be briefly introduced, the object modelling technique (OMT).

OMT was originally developed by James Rumbaugh[18] at General Electric as a series of steps for the production of software systems, with associated techniques and notations and

claims to be suitable for full life-cycle development and for development using rapid prototyping. The methodology has three main stages; analysis, design and implementation, but strongly emphasises the idea of a seamless development process. Central to this process is a model of objects in the problem domain to which design and implementation details are added during development. Rumbaugh's version of the object model has been widely used in object-oriented system development and is one of the principal components of the Unified Modelling Language (UML).

One of the difficulties arising from the popularity of the object-oriented approach to development systems is a lack of standardisation. Although the underlying concepts remain the same, there is a huge variation in notation and process among the object-oriented methodologies. The Unified Modelling Language aims to address this problem by combining the best elements of the main object-oriented methodologies while, at the same time, reflecting best practices in industry. The detail of using UML in object modelling along with system and object design is beyond the scope of the thesis[†].

4.2.3 Choosing a Methodology

The way in which a software system is developed has a huge influence on the final software product and can make the difference between success and failure in a development project. With regard to a choice between process-oriented and object-oriented methodologies, there were no immediately apparent reasons for choosing one over the other. Both allowed for the top-down design of the system in an implementation independent manner, with the ability to define the flows of information and control between either processes or objects.

Having considered both methods in terms of electronic system design and not found a clear superiority of one over the other, it was decided to base the selection upon the programming language used in the implementation of the system. It was considered that a Visual Programming language such as Visual Basic would be appropriate and specifically Version 6. Although Visual Basic 6 has object-oriented features, it is not a fully object-oriented language such as Java and VB.NET. Therefore, it was preferred not to use the object-oriented method to model software implemented in Visual Basic 6. This led to the adoption of the process-oriented methodology to which the form-based structure of Visual Basic was well suited. Specifically, each process in the Data Flow Diagram can be translated into a 'form' in a Visual Basic project.

[†] Further information be found in '*Object-Oriented Software Engineering: Conquering Complex & Changing Systems*' by Bruegge & Dutoit[19].

4.3 Functional Decomposition

The functional decomposition of the filter design system used Data Flow Diagrams, the first stage of which is to establish the 'Statement of Purpose'. For this system the statement of purpose is[†]:

> *The system interacts with the users, whether expert or novice, to set their design needs and provides help and advice to the users upon request. The system can be linked to external design tools, where the user can set the design parameters. The system can search for a design case that meets the user request. The system assists the user in accepting or modifying a design and will store it as a new design in the system database.*

4.3.1 The Context Diagram

In the context diagram, the filter design system is represented by a single process as shown in Fig. 4.6 where the external entities are the user, design tools that can be used to assist in filter design and simulation and the 'Completed Design'. The flows between the external entities and the system are defined and entered in the data dictionary.

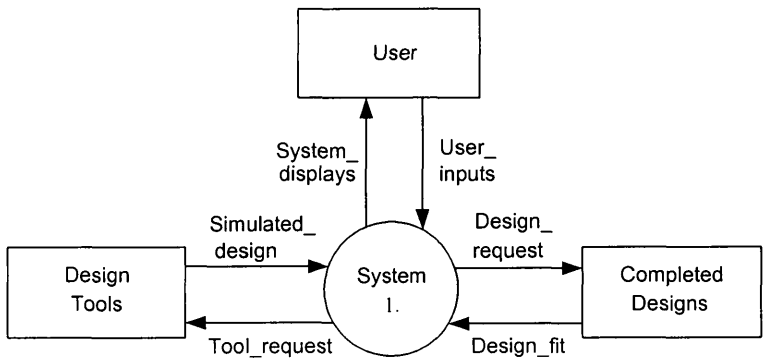


Figure 4.6: *Context diagram for filter design system*

4.3.2 Decomposing the Diagram

Having produced the context diagram of the filter design system and defined the relevant external entities, it is necessary to establish the internal structure of the system, which will result in the development of a full set of Data Flow Diagrams.

Process modelling allows for the design of the system to progress directly to the functional decomposition of the design using Data Flow Diagrams and there are several ways in which

[†] See also section 4.2.1

the Data Flow Diagram can be created[20]. One way, as suggested by Yourdon[21], is by developing an Event List for the system and then to use this as a starting point from which to construct the Data Flow Diagrams.

A second technique, which developed from the Event List, is to create Use Cases[22] as the first step in developing the Data Flow Diagrams. A Use Case is a set of activities that the system performs to produce some output result and is simpler in format than Data Flow Diagrams, and hence easier to understand. Each Use Case describes how the system reacts to an event that triggers the system.

The third approach is to skip both the Event List and Use Case and move directly to creating Data Flow Diagrams by directly thinking about system scenarios relating to the major systems functions.

As the proposed system is based on its functionality and interactions with the users and the external links, the third approach based on system scenarios was considered the most suitable and is adopted here. For the filter design system, the scenarios correspond to actions such as:

1. Choice of filter design by type and technology.
2. Choice of filter design by application.
3. Designing the filter by setting of specifications.
4. Optimising the filter design by setting design criteria.
5. Searching for the desired filter design.
6. Modifying the existing design to a new design.

The above steps represent the high-level functions of the system represented by the single process in the context diagram of Fig 4.6, and also derive from the earlier simplified diagram of the filter design expert system (Fig. 3.9, Section 3.4). These steps can now be represented by the six main processes forming the first level of Data Flow Diagram as shown in Fig. 4.7.

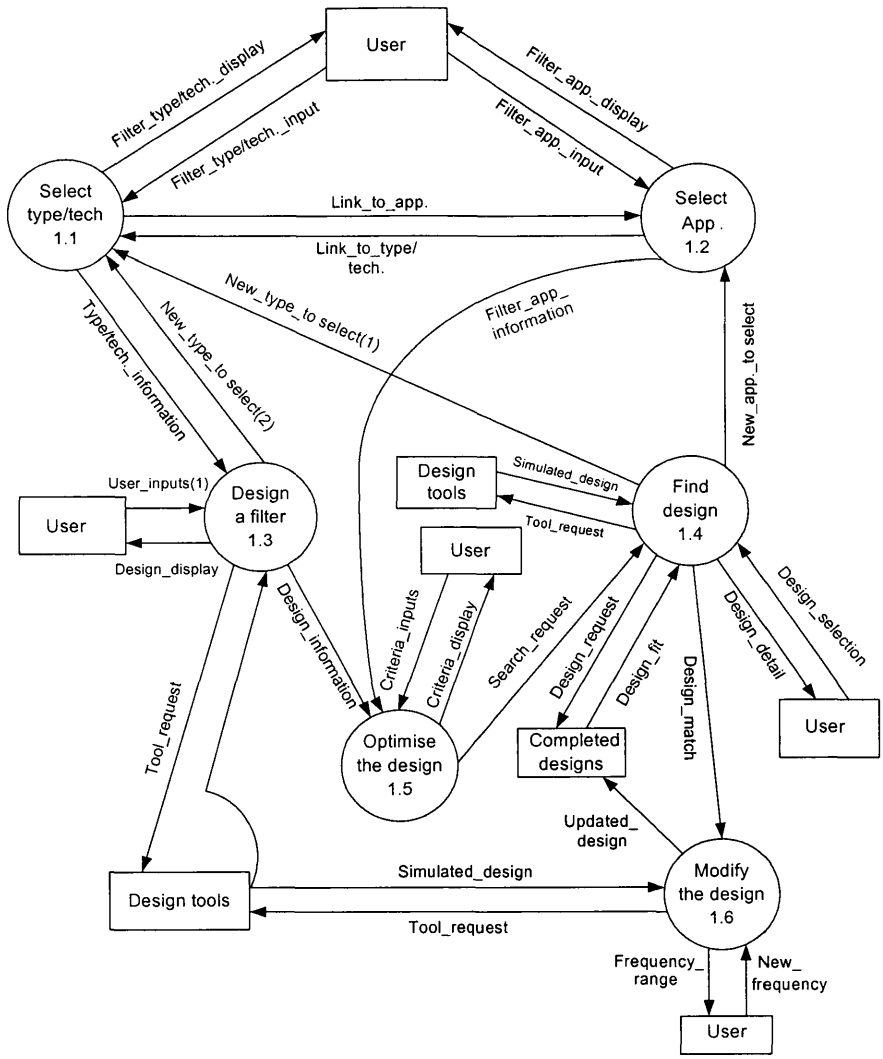


Figure 4.7: *First level decomposition of the filter design system*

The external entities in the context diagram are the same as in this diagram; namely the user, design tools, and completed designs. The first level diagram shows all the major high level processes and how they are interrelated. The flows between common points were grouped; individual flows are defined and entered in the data dictionary. These are then grouped into a single flow in each direction for clarity.

To ease understanding of the operations of Fig 4.7, the data flows are summarised in Table 4.1.

Table 4.1: *Flow descriptions for the first level DFD of the filter design system*

Flow	Source	Destination	Data/Function
Design_fit	Completed designs	Process 1.4	Information about the design fit
Simulated_design	Design tools	Process 1.3, 1.6	Information of the design simulation
Link_to_app	Process 1.1	Process 1.2	Link from type/tech to App. process
Type/tech._information	Process 1.1	Process 1.3	Information of filter type/tech.
Filter_type/tech._display	Process 1.1	User	Display type/tech. options
Link_to_type/tech.	Process 1.2	Process 1.1	Link from the App to type/tech. process
Filter_app._info.	Process 1.2	Process 1.5	Information of filter application
Filter_app_display	Process 1.2	User	The display of filter applications
Tool_request	Process 1.3	Design tools	Links the system to external design applications
New_type_to_select	Process 1.3	Process 1.1	Link to process 1.1 for new type/tech. design
Design_information	Process 1.3	Process 1.5	Design information to be optimised
Design_display	Process 1.3	User	Flows included design_parameters_display of all types.
Design_request	Process 1.4	Completed designs	To request design from database
New_type_to_select(1)	Process 1.4	Process 1.1	Link to process 1.1 for new type/tech. design
New_app._to select	Process 1.4	Process 1.2	Link to process 1.2 to select new filter application
Design_match	Process 1.4	Process 1.6	To display the searched filter design detail
Design_detail	Process 1.4	User	Information of the displayed design
Search_request	Process 1.5	Process 1.4	Links to database file to search
Criteria_display	Process 1.5	User	To display the optimising design criteria
Updated_design	Process 1.6	Completed design	Information of the new modified design
Frequency_range	Process 1.6	User	Information to display frequency range
Filter_type/tech. input	User	Process 1.1	Information on filter type/tech.
Filter_app._input	User	Process 1.2	Information of filter application set by the user
User_inputs(1)	User	Process 1.3	Flows included design_parameters_input of all types.
Design_selection	User	Process 1.4	Information to select the appropriate retrieval design
Criteria_inputs	User	Process 1.5	Information for the required criteria
New_frequency	User	Process 1.6	Information of new frequency of the design

The processes are:

Process 1.1 *"Select type/tech":*

This establishes the user choice by first identifying the type and technology of the filter. Choosing this process as the first process in the decomposition was based on the filter hierarchy tree established in Chapter 3, in which type and technology were positioned at the top. The data flows connected to this process depend on their source and their destination. For instance, the output flow from the process to the "user" terminator represents a list of filter types and technology from which the user can select.

Process 1.2 *"Select application":*

This represents the second major function to be achieved by the system, namely finding a filter design fit by choosing its application. This process represents the short cut for a user looking for a quick design solution based on a particular filter application. Referring to the

diagram of Fig.4.7, there is an output flow from this process to the "user" terminator, which has information on filter applications whereby the user can select the required one and feed it back to the process.

There is also an outgoing flow from this process that has information on filter applications to process 1.4, where the procedure for finding a design is begun.

Process 1.3 *"Design a filter":*
This process represents the filter design process for a customised design. This will involve filter type, filter technology and other design parameters such as filter response, cut-off frequency, ripple and so forth. There are incoming and outgoing flows from this process to process 1.1 as presented in Table 4.1. Other flows are to and from the "user" terminator, enabling the user to select from the list of filter design parameters to complete the custom design. User selection is connected to process 1.4 "*Design a filter*" through the outgoing flow "*Design_information*".

At this stage, it is important for the user to have access to a filter design tool that will enable them to view the expected filter response to ensure that it meets the specification. Such a link is represented in the diagram by the outgoing flow from process 1.3 as "*Tool_request*", and the incoming flow to the process is in a form "*Simulated_design*". The user can try different design parameters until the required performance is achieved.

Process 1.4 *"Find a design":*
Having selected the required filter design either by filter application or by customising the filter design and having used the links to the design tools to simulate the design, this is where the system starts looking for a design match. The incoming flow from process 1.5 is the request for a design search, which acts after the optimising criteria were set in the optimising process. The outgoing flow from this process to the user is a list of the returned designs from which the user then can select the required design. The existing and the new designs are stored in "*Completed designs*", which is linked to process 1.4 through the outgoing flow "*Design_request*" and the incoming flow "*Design_fit*". There are also flows to and from this process to the terminator "*Design tools*".

Another flow from this process occurs when the user retrieves an existed design but is not fully satisfied with the match. The user can then opt to modify the design. This is done through the outgoing flow "*Design_match*" to process 1.6 "*Modify the design*". If there is a need for a new solution, for instance if there is no acceptable match or the user decides to select a new filter type, then this would be represented by the outgoing flow "*New type to select(1)*" to process 1.1.

Process 1.5 *"Optimise the design"*:

This is where the user sets the design optimisation criteria for the filter by either selecting the filter application or the custom design processes. The outgoing flow acts to display a list of design criteria, whereas the incoming flow is the optimising criteria that have been selected.

Process 1.6 *"Modify the design"*:

The user may not be fully satisfied with the returned design matches, in which case this process will represent the input and the output flows necessary to modify the design. For instance, if there is a need to change the retrieved filter design cut-off frequency, then to modify the design there is a need to use filter design tools where the user has the freedom to modify the design to achieve the required frequency response. Having modified the design, it can now be stored as a new design. This is represented in the diagram by the outgoing flow "*Updated_design*" from process 1.6 to the terminator "*Completed designs*".

4.3.3 The Data Dictionary

The relationship between the Data Dictionary and the Data Flow Diagram is shown in Fig. 4.8, while Fig 4.9 provides an example of the format used.

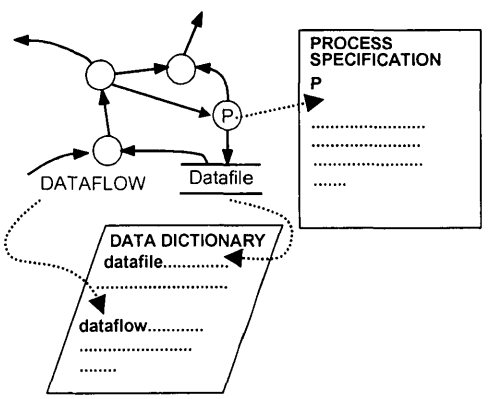


Figure 4.8: *Relationship between Data Dictionary, Process Specification and the Data Flow Diagram*

Data Dictionary Entry.
Name: Filter type input
Form: Text
Description: Sets filter type i.e. lowpass, highpass, bandpass or bandstop

Figure 4.9: *Data Dictionary entry*

The Data Dictionary as set out in Fig. 4.9 contains three elements; the '*Name*' field, the '*Form*' field and the description. The *Name* field specifies the flow or group of flows to which the Data Dictionary entry corresponds. The *Form* field identifies, where appropriate, the variable, which is to be used as the information carrier for the flow.

It was recognised that the *Form* field would be required to be flexible in its format, particularly as it was considered desirable for the means of information transfer along different flows to be defined at different levels of abstraction depending on the way in which the flow was defined. This would take into account external considerations such as the fact that some forms of flows are constrained by their type. For instance, the flow '*Tool_request*' in Fig. 4.7 is in the form of a tool link, which can be represented by a command button or an icon or a function key or can be accessed through the menu. Therefore, the form of this flow was externally constrained by the type of link used to provide the requisite function. Therefore it is necessary to consider the form field of a flow from the earliest stage to ensure that any subsequent design decisions made, which require interaction with a flow, would be made aware of its form, ensuring that any interfacing processes provide the appropriate termination.

The final part of the Data Dictionary in Fig 4.9 is the description of the flow, which is provided for the benefit of the designer and is intended as being purely descriptive.

The combined flows that are shown on the context diagram in Fig. 4.6, termed here "*Compound flows*" are defined for clarity only in the higher level diagrams, where many flows are connected between common nodes. All flows in the same direction between the nodes are grouped into a single flow and given a group name. Since the flow does not actually exist as far as the implementation of the design is concerned in that it does not of itself provide either control or information to the system, there is no need to define the form of the flow or define its purpose as with normal flows.

The compound flow for those flows originating at the 'User' and terminating at the 'system' and including the '*Filter type input*' flow defined above, is shown in Fig. 4.10 below:

Data Dictionary Entry
Name: User inputs
Flows included: Filter_type_input; Filter_tech._Input; Filter_application_input; User_inputs; Criteria_inputs; Design_selection; New_frequency.

Figure 4.10: *Data Dictionary for the compound flow " User_inputs"*

The full Data Dictionary is presented, together with a full set of data flow diagrams, in Appendix A.

4.4 Completing the Functional Decomposition

Having developed the first level of decomposition from the context diagram based on a consideration of the system major functions, it was necessary to continue the decomposition to increase the level of detail to the point where the individual processes cannot reasonably be decomposed further. This creates a set of hierarchically related charts in which a process on any given chart may be described in greater detail on another, lower level, chart.

The way in which individual charts are numbered offers the additional benefit of assuring that neighbouring processes, which if the decomposition process has been undertaken correctly are also those most closely related, are considered together.

To see how the first level of the Data Flow Diagram can be further decomposed, consider process 1.1 *"Select type/tech"* in Fig. 4.7.

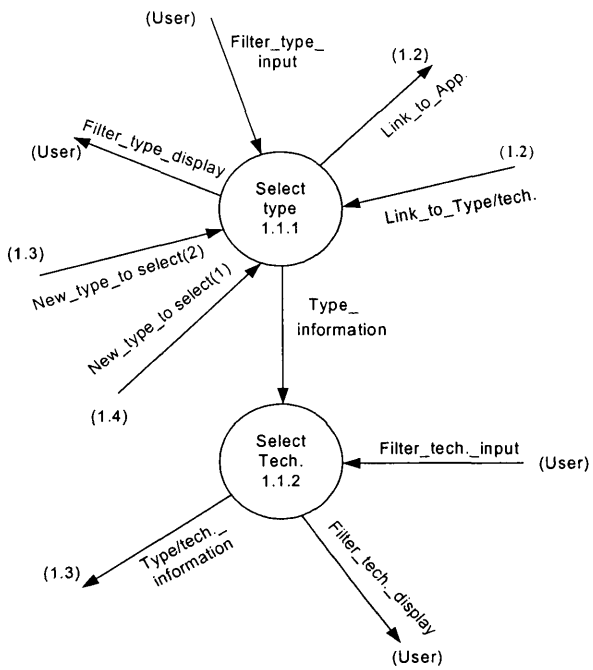


Figure 4.11: *The decomposition of process 1.1*

This process communicates with the user to display the filter types and technologies available for implementation and enables the user to make a choice. It also communicates with process 1.2 *"Select Application"* and process 1.3 *"Design a Filter"* and receives information from process 1.4 *"Find Design"*. Decomposing process 1.1 results in the Data Flow Diagram of Fig. 4.11, which contains two new processes, process 1.1.1 to perform the *"Select type"* operation and process 1.1.2 to perform the *"Select technology"* operation.

Process 1.3 in Fig 4.7 may be considered as the heart of the overall filter design process. Process 1.3 can be decomposed as shown in Fig. 4.12, which contains seven lower level processes associated with the different filter technologies. From Fig. 4.12, it can be seen that process 1.3 is decomposed into processes associated with the filter type and technology, including the setting of design parameters and the displaying of the designed filter response to the user.

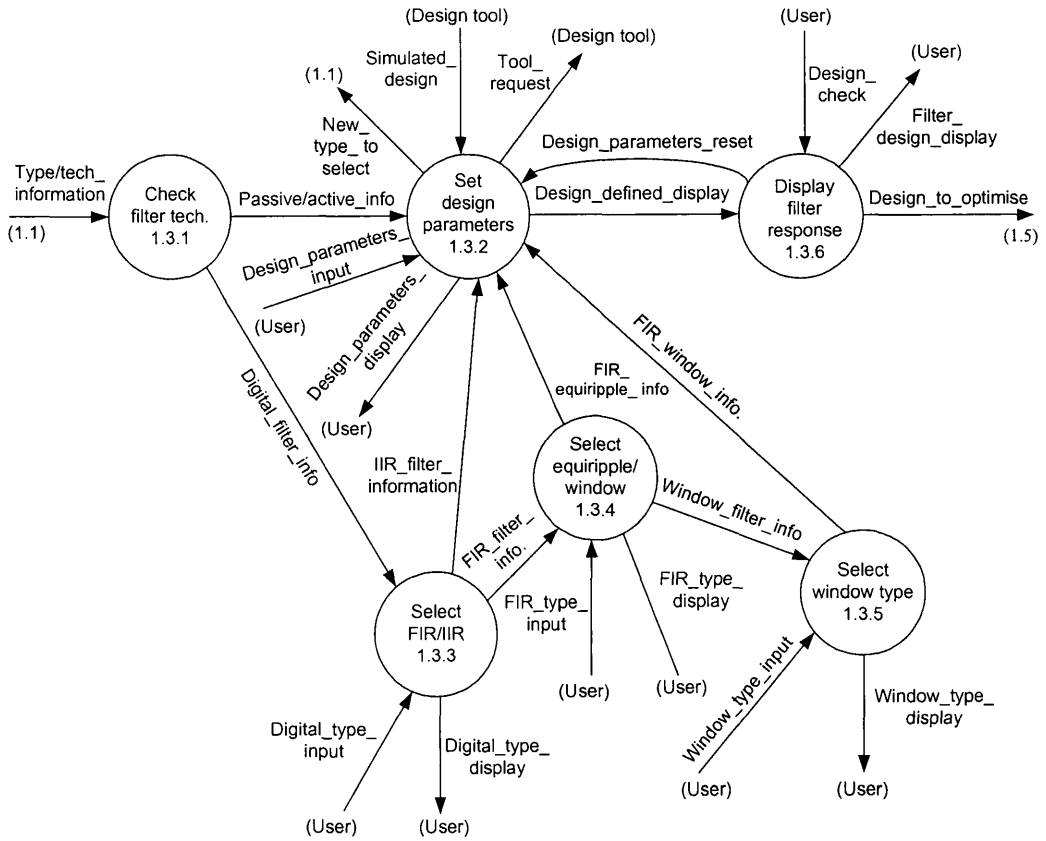


Figure 4.12: *Decomposition of process 1.3*

The decomposition of the other processes in the main Data Flow Diagram is shown in Appendix A. From the decomposition of the level 1 diagram by one further level it was determined that enough detail had been achieved for the purpose of developing the filter design support tool and that no further decomposition was needed.

Having decomposed all processes to the point where no further decomposition is needed, it is now necessary to represent each low level process using Structure English. The Structure English form can then be used as a pseudo code which the programmer can use in implementing the system.

Consider process 1.1.1, the *Select type* process shown in Fig. 4.11. A Structured English description of the function of the process might be as follows:

Process Specification: *Select Type*

BEGIN

IF *Filter_ type_display* = 'ON'
THEN Select *filter_ type_ input*

IF filter_type_input = 'ON'
THEN GOTO *Select tech.* Process
ELSE GOTO select *App. Process*

IF *New_ type_to select(1)* = 'ON'
AND *New_type_to select(2)* = 'ON'
OR *Link_to_type/app.* = 'ON'
THEN *Filter_ type_ display* = 'ON'

END

The same consideration is given to each process. A full list of process specifications developed is given in Appendix A.

4.5 System Design

In the previous sections of this Chapter, the process of system analysis was performed using Data Flow Diagrams and logic modelling to represent the internal structure and functionality of the relevant processes.

This section is concerned with the system design or how to build the system, not system analysis, which deals with establishing needs. During the initial part of design, the diagrams from the analysis phase convert into physical diagrams that explain how to build the system. All the documentation and diagrams developed in the analysis serve as input to the design phase, either directly or with modifications.

The system design firstly involves the logical design that defines the 'look and feel' of all system inputs, outputs, interfaces and dialogues. Secondly, the physical design which involves specifying all the technological characteristics of the system so that those involved in the implementation phase can concentrate on building the system, and including the designing of the physical files and databases along with the system and program structures.

For example, Data Flow Diagrams are converted into diagrams that show the implementation details and explain how the final system works. Structured English is turned into pseudo code, which is then used by the programmer to develop the high level code in which the system is implemented.

The process of designing interfaces and dialogues is a user-focused activity. This means that it follows a prototyping methodology of interactively collecting information, construction a prototype, assessing usability, and making refinements. To design usable interfaces and

associated dialogues, the '*who, what, where, why and how*' questions related to the creation of all the system interfaces or dialogues must be answered. Gaining understanding of these questions is therefore a necessary first step in the creation of any interface or dialogue.

For example, an understanding of the nature of the users, their skills and abilities will greatly enhance the creation of an effective design. In other words:

- Are the users experienced computer users or novices?
- What are their educational levels, background, and task- relevant knowledge?

Answers to these and other related questions would provide guidance for both the format and content of the designs. For instance, the users are assumed to:

- Have a mechatronics background.
- Are at least of degree level.
- They are familiar with computers.
- They may be either experienced in or a novice at filter design.

The design will be based on the system specification and the design scenarios, which will represent the range of users.

Also:

- What is the purpose of the interface or dialogue?
- What tasks will the users be performing?
- What information needed to complete this task?

Other questions are:

- Where will the users be when performing their task?
- Will users have access to online systems or will they be in the field?
- What sort of help the will a user need from the system?

After establishing the initial requirements, this information is used to generate a prototype structure and to refine the requirements independently of the users, although users will be asked to review and evaluate the prototype.

After reviewing the prototype, users may accept the design or request that changes are to be made. If changes are needed, the construction-evaluate-refine cycle is repeated until the design is accepted. Usually, several iterations of this cycle occur during the design of a single form or window. As with any prototyping process, these iterations should occur as rapidly as possible in order to gain the greatest benefits from this design approach.

4.5.1 Designing the System Interface

In this section, the interface layout will be designed. This provides the guidelines for structuring and controlling the data entry fields, providing feedback and designing the on-line help. Effective interface design requires a thorough understanding of each of these concepts.

Layout Design:

Most software designed for personal computers follows the standard Windows or Macintosh approach in which the screen is divided into a number of boxes, each of which represents a navigation area through which the user issues commands to navigate through the system.

A concern when designing the layout of computer-based forms is the requirement for field navigation. Following the sequence which users typically use to control moves between fields, standard screen navigation flows from left to right and top to bottom, and this will be followed here.

Depending upon the application, various functional capabilities will be required to provide smooth navigation and data entry. For example, a consistent interface will provide common ways for users to move to different places on the form, to edit characters and fields, to move among form displays and obtain help. Keystrokes, mouse or other pointing device operations, menu selection or button activation may provide any or all of these functions.

The interface for the filter design expert system will be based on forms, allowing users to fill in the blanks when working with the system. Form interaction is defined by Hoffer[23] as a highly intuitive human-computer interaction method whereby data fields are formatted in a manner similar to paper-based forms and is effective both for input and the presentation of information. An effectively designed form includes title and field headings, provides default values when practical, and displays data at an appropriate field length.

The easiest way to design the form layout is first to generate paper-based forms and then to move to the equivalent computer based form. The forms comprise elements such as form header, text labels, a text boxes to enter data, combo boxes to select data, command buttons to lead users, option buttons for selection, data display and message boxes. Consider the processes 1.2, 1.4 and 1.5 in the first level Data Flow Diagram along with their lower levels decomposition in appendix A2 these can be represented by the computer based form shown in Fig.4.13. This form enables a user to search for a filter design match by selecting filter application and setting design criteria.

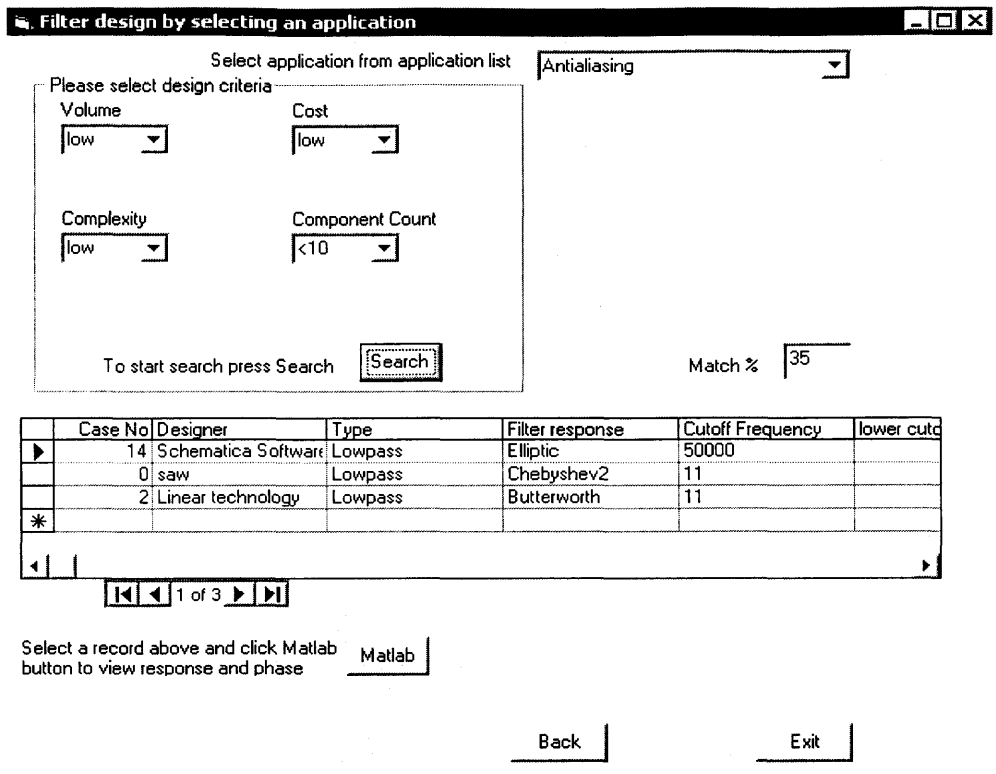


Figure 4.13: *Computer based form of filter design by selecting filter application*

Structured Data Entry:

There are a number of rules to be considered when structuring the data entry fields on a form[24]. To minimise data entry errors and user frustration, the user should never be required to enter information which is already available within the system or that can be easily computed by the system. For instance, never require the user to enter the current date and time, since these values can easily be retrieved by the computer.

Similarly, always provide default values when appropriate. For example, when the user first enters the filter design form, two fields are set to default values; the filter response is set to Butterworth and the filter order set to the minimum filter order, namely 2. The use of default values will reduce errors and ease and speed up the processing of the application.

When entering data, the user should not be worried about the dimensional units of a particular value. For example, in the design form, the user can enter the cut-off frequency in either Hertz or rad/s. Field formatting and data entry prompt should make clear the type of data being requested. In other words, a caption describing the data to be entered or selected should be adjacent to each data field. Within this caption, it should be clear to the user what type of data is being requested.

In order to reduce errors when entering new data, forms were designed to minimise data entry by using data selection from lists. Figure 4.14 shows a combo box listing the filter design applications. The user can select the required one from the list by clicking the mouse.

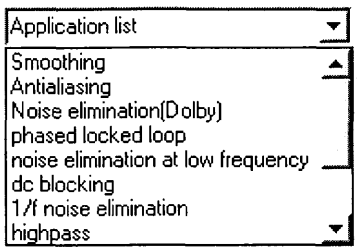


Figure 4.14: *A combo box for filter design application list*

Providing Feedback:

When designing system interfaces, providing appropriate feedback is a means of making a user's interaction more effective. Not providing feedback is generally a way to frustrate and confuse. Feedback is typically of one of three types:

1. Status information
2. Prompting cues
3. Error or warning messages

Status information - This is a simple technique for keeping users informed of what is going on within the system, for example, displaying a splash screen.

Placing appropriate information on each screen also provides feedback to the user. Thus when opening a file it might display a message such as "*Please wait while the file is opening*" or when performing a large calculation or a search, flash the message "*Working...*" or show a progress bar to the user.

Error or warning messages – These messages should be specific and free of error codes and jargon. Additionally, messages should never scold the user and should attempt to guide them toward a resolution. For example, a message might say "*Please enter cut-off frequency between 1 and 100000 rad/s*" and the message title "*Frequency Limit*" as a warning message to the user if the entered frequency was out of range, see Fig. 4.15.

Messages should be in user, not computer, terms. Hence, terms such as "*End of File*", "*Disk I/O error*", or "*Write Protected*" may be too technical and not helpful for many users.

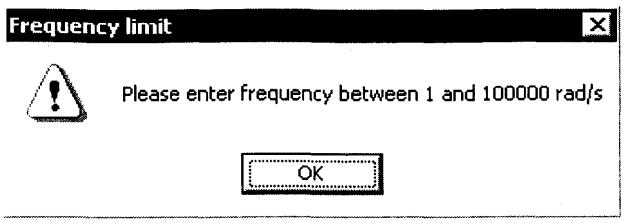


Figure 4.15: *Frequency limit message*

Providing help - Determining how to provide help is one of the most important interface design issues. When providing help it needs to be looked at from the user's perspective. When accessing help, the user likely does not know what to do next, does not understand what is being requested, or does not know how the requested information needs to be formatted. In order to design a usable help, the first message should be short, to the point, and use words that enable understanding.

This leads to a second guideline, that of organisation, which means that help messages should be written so that users can easily absorb information. For instance, the organisation of information through the use of bulleted and ordered lists may help the user. Finally, it is often useful to explicitly show users how to perform an operation and the outcomes of associated procedural steps.

Many commercially available systems provide extensive on-line system help. Many systems are also designed so that users can vary the level of help detail provided. Help may be provided at the system level, screen or form level as well as at the level of an individual field. The ability to provide field level help is often referred to as "*context-sensitive*" help. For some applications providing context-sensitive help for all system options is an undertaking that is virtually a project in itself.

If it is decided to design an extensive help system with many levels of detail, it must be established exactly what the user needs help with or the efforts may confuse users more than help them. After leaving help, users should always be returned back to where they were prior to requesting help. If these simple guidelines were followed, a highly usable help system should be developed.

4.5.2 Designing Dialogues

The process of designing the overall sequence that users follow to interact with an information system is called dialogue design. The design rule is to select the most appropriate interaction methods and devices, and define conditions under which information is displayed and obtained from users. These are a few general rules that should be followed when designing a dialogue and these are summarised in Table 4.2. For a dialogue to have

high usability, it must be consistent in form, function, and style. All other rules regarding dialogue design are mitigated by the consistency guideline. For example, the effectiveness of how well errors are handled or feedback is provided will be significantly influenced by consistency in design.

Table 4.2: *Guidelines for the design of dialogues*[25]

Guideline	*Explanation*
Consistency	Dialogues should be consistent in sequence of actions, keystrokes, and terminology.
Shortcuts and sequence	Allow advanced users to take shortcuts using special keys (CTRL-C to copy). A natural sequence of steps should be followed (enter first name before last name).
Feedback	Feedback should be provided for every user action (confirm that a record has been added, rather than simply putting another blank form on the screen)
Closure	Dialogues should be logically grouped and have a beginning, middle, and end (the last in the sequence of screens should indicate that no more screens.
Reversal	Dialogues should, when possible, allow the user to reverse actions (undo a deletion); data should not be destructed without confirmation.
Ease	Dialogue should be simple to enter information and navigate between screens (Use forward, backward, and move to specific screens such as first and last).

Designing the Dialogue Sequence:

The first step in dialogue design is to define the sequence. In other words, it must be understood how users might interact with the system. This means that there must be a clear understanding of user, task, and technological and environmental characteristics when designing dialogues, requiring that the system specification and analysis must be reviewed. The Data Flow Diagrams, data element list and the design of the system output were all considered and as a result the following dialogue flow for the filter design expert system was established:

1. Set the design based on either user specification or filter application.
2. Set user's specification such as filter type, response or frequency
3. Select filter application and associated criteria.
4. Search the database file for a design.
5. Accept design.
6. Modify and save the design.
7. Create a new design.

As a designer, the understanding of how a user wishes to use a system enables the transformation of these activities into a formal dialogue specification. A formal method for designing and representing dialogues is dialogue diagramming[26]. Dialogue diagrams have only one symbol, a box with three sections as shown in Fig. 4.16, each box then represents a specific form or window within a dialogue.

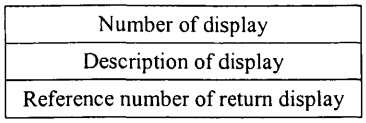


Figure 4.16: *Sections of a dialogue diagramming box*

These three sections of the box are used as follows:

Top - Contains a unique display reference number used by other displays for referencing it.

Middle - Contains the name or description of the display.

Bottom - Contains the display reference number accessed from the current display.

The dialogue diagram for accessing the filter design expert system is as shown in Fig. 4.17.

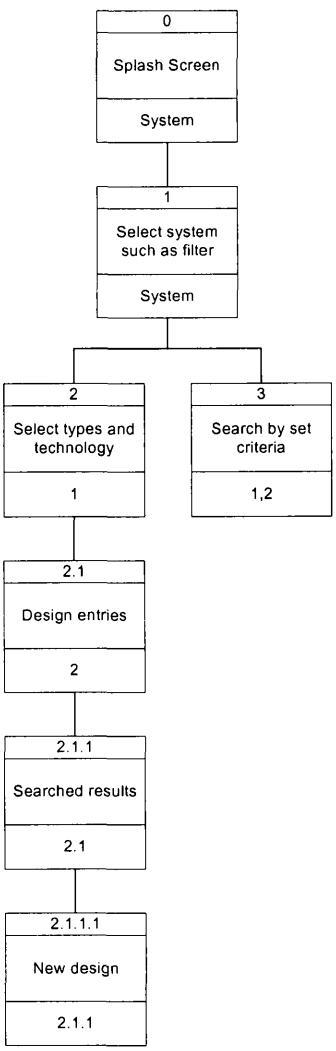


Figure 4.17: *Dialogue diagram for filter design expert system*

The links between the different items of the dialogue diagram and the processes in the first level Data Flow Diagram of Fig 4.7 are shown in Table 4.3. From this table it is clear that each item in the dialogue diagram is corresponding to the related process in terms of receiving and passing data.

Table 4.3: *Referring items to the corresponding processes in level 1, DFD.*

Item number	Item name	Related process
0	Splash screen	1.1
1	Select system	1.1
2	Select type/tech.	1.1
2.1	Design entries	1.3, 1.5
2.1.1	Searched results	1.4
2.1.1.1	New design	1.6
3	Search by filter application	1.2

From the diagram of Fig. 4.17, the user would first see a splash screen, which provides the user with the system information, and which from the implementation point of view provides a time delay while the system is loading. This box contains a reference number of "0" or (Item 0). A main menu (Item 1) is then displayed which has two options:

1. Selects the filter type and technology (Item 2)
2. Selecting filter requirements or application (Item 3)

When the user selects Item 2, control is transferred to the design entries display (Item 2.1). As the filter design entries are selected, the user is presented with a window with the search results relating to the required design (Item 2.1.1). At this stage the user can accept the design or modify it at that window or can go to another window for a new design (Item 2.1.1.1).

The other option, which can be selected at Item 1, is to search for a design by application and set criteria (Item 3). In this instance the user can search for a design by selecting a filter application and can then further refine the search by setting design criteria. The user can at the same window either accept the existed design or modify it, or can go to the window of select type and technology at Item 2 if a new design is desired. Fig. 4.18 shows an example of the screen equivalent of the box of Item 3; the whole interface involved in the implementation stage will be described in the next Chapter.

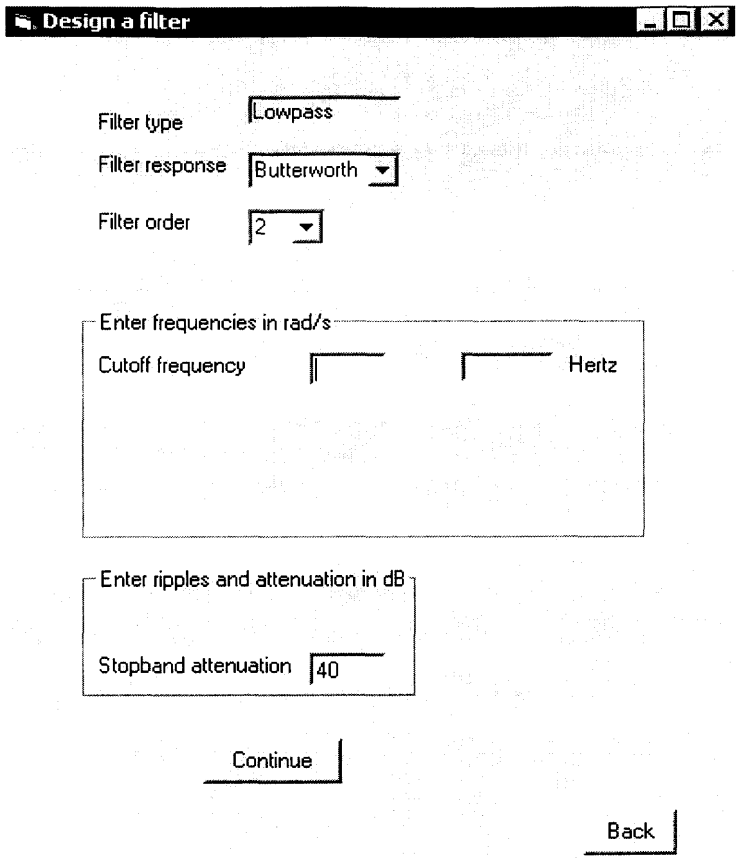


Figure 4.18: *Computer based form*

Building a Prototype and Assessing Usability:

Building a dialogue prototype for the filter design expert system can be relatively straight forward, especially when using CASE tools or any of the graphical development environments such as Visual Basic. Such activities are not only useful to show how an interface will look and feel, they are also useful for assessing usability and for performing user training long before actual systems are completed.

At this stage a complete look and feel application has been built but which is not functional, but shows all the interface structures in a form that a user can go through. An example of a look and feel form is as that shown earlier in Fig. 4.18. Such forms have no code underneath then to make operational, and therefore the only functional commands are those that link the forms to each other such as "Next" and "Back" buttons or the "Exit" button.

4.5.3 Data Storage Design

The data storage function manages how data is stored and handled by the programme. This section describes how the data storage for the project was designed using a two-step

approach, firstly selecting the format of the data storage and then optimising it to perform efficiently.

There are two main types of data storage formats available - files and databases. Files are electronic lists of data that have been optimised to perform a particular task. The information in the file is in the form in which it is used so that it can be accessed and processed quickly by the system. A database is a collection of groupings of information that are related to each other through common fields. Logical grouping of information could include categories such as filter type, filter response and cut-off frequency. A database management system (DBMS) is software that creates and manipulates the database. An end-user DBMS such as Microsoft Access supports small scale databases that are used to enhance personal productivity.

Selecting a Storage Format:

In order to select the appropriate storage format for this project, it is necessary to undertake a comparison of data storage formats and their impact on the design and implementation of the project. Generally, each of the file and database data formats has its individual strengths and weaknesses, and no format is inherently better than the others.

In fact, sometimes a design can choose to use multiple data storage formats. For instance, a database for one data store, a file for another. The selection here will be based on task-dependency and the way in which the choice will serve the design and implementation of the system. The first issue that can help identify the appropriate file or database format is that of data type. Most applications need to store simple data types; such as text, dates, and numbers, and all DBMS are equipped to handle this kind of data as this can be done using fields. However, the best choice for simple data storage is usually the database where technology has matured and improved over time to handle simple data very effectively. Some applications incorporate data such as circuit diagrams and images, which can also be handled easily by the database. Other advantages in using a database arise from the implementation perspective where a consistency was found between the programming language used and the database. Microsoft Visual Basic was used to design the user front end, which can be linked easily to the Microsoft Access database and to the Structured Query Language (SQL). The use of a database will therefore ease the design and make the implementation more efficient.

When translating the filter design cases into records in a database, these cases fitted conveniently in a single table, therefore SQL statements are used to identify a specific filter design in the table in which the individual records are held. Each record represents a filter design case and consists of several fields or attributes.

Fields:

A field is the smallest unit of application data recognised by system software. In the case of the filter design database each entry contains 20 fields. At this stage, when designing the fields, the data type must be considered. The data type of the filter design case fields varies from number, text and note (memo), see Fig. 4.19. The field size also can be set by selecting the appropriate data type for each field and will have the benefit of minimise storage space, represent all possible values and supporting data manipulation.

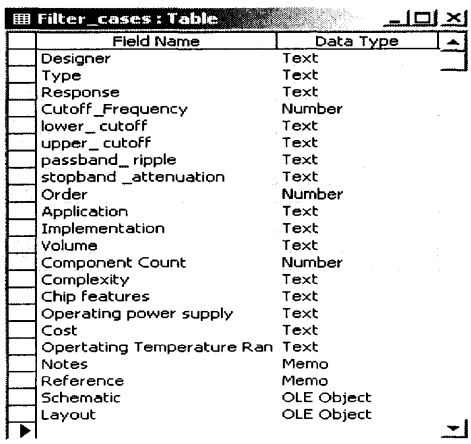


Fig. 4.19: *The data type for filter design fields*

Designing Records:

A record is a group of fields stored in adjacent memory locations in a file or a database table and retrieved together as a unit. The individual filter design cases can be modelled into records. Initially there are about 12 filter design cases and these are stored as individual records in the database table.

The design of records involves choosing the sequence and grouping of fields into adjacent storage locations to achieve efficient storage and data processing speed. The fields in the filter design records are entered in the following sequence:

- Case number.
- Designer name.
- Filter type.
- Filter response.

These are then followed by the fields involving filter specification, including:

- Cut-off frequency.
- Filter order.
- Ripple.

Next are the fields associated with design optimisation such as:

- Cost.
- Volume.
- Complexity.
- Component count.

Finally there are fields associated with items such as:

- Design notes.
- Application.
- Chip features.
- Power supply.
- Temperature.
- Source reference for the design.

4.5.4 Program Design

Program design is that part of the design phase of the system development life cycle during which analysts create instructions for the programmers as to how the code should be written and how pieces of code should fit together to form a program. The program design techniques are important because it will ease the programmer's job in understanding and organising the program and in producing the original code that supports the application logic of the system.

It is possible to go to the implementation phase without much thought or planning, but this can lead to unwanted results such as inefficient programs, code that does not work with other code and a system that does not do what it is supposed to do. Instead, analysts should first take time in the design phase to create a maintainable system based on a design that is modular and flexible. To do this, analysts can design programs using a top-down, modular approach, deploying a variety of program design techniques.

Generally, the process begins with a high-level diagram that shows the various components of a program, how these components should be organised, and how components interrelate. This diagram, known as the structure chart, illustrates the organisation and interactions of the different pieces of code within the program that programmers can develop the program independently.

Process modules are used as the starting point for structure charts. Each process of the Data Flow Diagram (DFD) tends to represent one module on the structure chart, and if levelled DFDs are used, then each DFD level trends to correspond to a different level of the structure

chart hierarchy. For example, the process on the context-level DFD would correspond to the top module on the structure chart.

The reason for converting a Data Flow Diagram into a structure chart when designing a program is because the structure chart shows all the components of code that must be included in a program at a high level, arranged in a hierarchical format that implies sequence, selection and iteration of components.

Once the overall program is defined at a high level, program specifications are written to describe exactly what needs to be included in each program module. The specification includes basic module information, for example, its name, calculations that need to be performed, and the target programming language. It would typically also include special code using a technique similar to Structured English to communicate what needs to be written by using programming structures and a generic language that is not program language specific.

When the analyst and programmer are the same person, then there will be no communication problems. In this project the pseudo code for the system can be developed directly from the Structured English of each process, which will lead to the development of the final code for the implementation.

The Data Flow Diagrams created at the beginning of this Chapter represented all the processes at different design levels. The process specification was derived for each process using Structured English. An example of the Structured English description for a process was given in Sec. 4.4.

The Structured English for each process can be further developed by adding more details of program development, equivalent to the details in pseudo code. This task can easily be done here, as the analyst and programmer are the same person.

Thus, the Structured English that describes the process of the filter design expert system, and which is shown in Appendix A, can be developed directly into pseudo code. The Data Flow Diagram (DFD) descriptions in Structured English are used as the primary input in producing this pseudo code. Pseudo code is a language that contains logical structure, including sequential statements, conditional statements and iteration. It differs from Structured English in that it contains details that are programming specific, such as initialisation instructions or linking. It is also more extensive so that the programmer can write the module by mirroring the pseudo code instructions.

In general, pseudo code is much more like real code, and for that reason it was decided that when converting the Structured English to pseudo code to carry on to convert it into the high

level code. This is because the high level code is not very different from the pseudo code and it was found to be convenient to translate the Structured English descriptions into the Visual Basic code.

4.6 Summary

Formal software methods were introduced and the process method chosen as more appropriate to modelling the system. Following the system process modelling using Data Flow Diagrams, the functional decomposition continued until the system could not be decomposed further. At the same time, the entries for the Data Dictionary of each flow and the details of the Process Specification for each process were developed. The design for the system was split into two design categories; the logical design, which mainly involved the design of system forms, interfaces and dialogues and the physical design, including the design of the physical files and database and program modules. This led to the development of the pseudo code for each of the program modules, which is converted into the code to implement the system.

The design stages presented in this Chapter provide an entry to the system implementation discussed in the next Chapter where Visual Basic, along with Structured Query Language and a Database Management System, is used to implement the different parts of the system design. The implementation of a system prototype including the database implementation and the program code will also be discussed in the next Chapter.

---

[1] Hoffer, J. A., George, J. F. and Valacich, J. S. (1999). *Modern System Analysis and Design*. USA: Addison Wesley Longman, Inc.

[2] DeMarco, T. (1979). *Structured Analysis and System Specification,* Englewood Cliffs, NJ: prentice-Hall

[3] Yourdon, E. and Constantine L.L. (1979). *Structured Design.* Englewood Cliffs, NJ: Prentice-Hall.

[4] Gane, C. and Sarson T. (1979). *Structured System Analysis.* Englewood Cliffs, NJ: Prentice-Hall.

[5] Yourdon E. (1989), *Modern Structured Analysis,* Englewood Cliffs, NJ, USA: Prentice-Hall,

[6] ibid., P339

[7] ibid., P160

[8] op.cit.1,P288

[9] op.cit. 3, P160

[10] op.cit. 1, P322

[11] op.cit. 5

[12] Bradley D., et al. (2000). *Mechatronics and the design of intelligent machines and systems*. UK: Stanley Thornes.

[13] Britton C, Oake J. (2000). *Object-Oriented System Development*. UK: McGraw-Hill Publishing Company.

[14] ibid., P14

[15] op. cit. 36, P195

[16] op.cit. 37,P23

[17] ibid. P38

[18] Rumbaugh, J., et al. (1991), *Object-Oriented Modelling and Design*. Englewood Cliffs, N.J: Prenetic-Hall.

[19] Brugge, B. and Dutoit, A.(2000). *Object-Oriented Software Engineering: Conquering Complex & Changing Systems*. Englewood Cliffs, N. J: Prentice Hall.

[20] Dennis a., Wixom B.H. (2000), System Analysis and design, John Wiley & Sons, USA

[21] op. cit.,5

[22] op.cit. 20, P153

[23] op. cit.,1

[24] ibid., P570

[25] op. cit. 1, 581

[26] ibid., 582

Chapter 5

System Implementation

5.1 Introduction

System implementation involves activities such as coding, testing, installation, documentation, training and support. However, as the main purpose of this research is to prove that a design support system as proposed is implementable, the emphasis here will be on coding and testing. The Chapter therefore discusses the conversion of the system designed in the previous Chapters into working and reliable software.

Coding is the process by which the design specifications are turned into working computer code. Once coding has begun, the testing process would begin and proceed in parallel. As each program module is produced, it can be tested individually, then as part of a larger program, and then as part of a larger system. Although the bulk of testing takes place during implementation, some testing has been done during the analysis phase described in Chapter 4. This involved the testing of the database file to check that each filter design case had complete design fields to ensure that the design is consistent, and ready to be searched by the inference engine. A further test involved checking each of the design cases and making sure that they matched the acquired filter design knowledge.

There are several languages that could be used to code the application. Among these are Visual Basic, C and Java. However, Microsoft Visual Basic has been chosen as this is suited to the type of the windows application development required. The Visual Basic environment makes it easy to build a user interface and it is also suited to building database applications involving Microsoft Access.

The first part of this Chapter will focus on the database design and the building of the physical database file. This is followed by the data access management and data control necessary to translate the user entries into the appropriate code to retrieve the required design case by searching the database file. This is based on the use of Structured Query Language (SQL) and other, related, Visual Basic code.

The second part of the Chapter is concerned with the testing of the application. The testing will involve the internal operations of the system modules to ensure that they perform according to specification together with high level or interface testing to ensure that the software performs as expected.

5.2 Using a Prototype

As mentioned in Chapter 2, an expert system is not built in a few large steps; rather it evolves toward a final form. This iterative approach to development is referred to as prototyping and it is an effective paradigm used for expert system development.

The filter design exemplar used in this research comprises three different filter types; passive, active and digital. There are similarities in the design characteristics of these types. For instance, passive and active filters are alike in their inputs and outputs with differences based on their physical circuits and specific applications. The required inputs and outputs are consistent across all filter types. Inputs are factors such as cut-off frequency; filter response, ripples and so forth and filter outputs include frequency response and phase[†]. Of course the internal structure of the filter differs between types, especially the digital filter which requires an analogue to digital (A/D) and digital to analogue (D/A) converters as well as a processor and memory.

There are similarities in the basic design principals across filter types. For instance, passive filters in terms of characteristic and operation are similar to active filters, although the passive filter is simpler and does not require a power supply. Active filters are however not as complex to build and use as digital filters, which, as indicated above, require both code and additional components. Therefore, it was decided to first build a prototype of the filter design expert system, based on an active filter.

By providing a mock-up of the application early on, it is possible to gauge the user's response and to react to user requirements before the expert system is too developed.

The first stage in creating the active filter prototype is to build a database file, which contains active filter design cases, then to build the user interface. Any form agreed upon can be coded and tested before adding it to other forms until the application is completed. This initial version, version 1, can then be developed further to create version 2 and so on, each of which is a refinement of the previous version. Once a robust system has been developed for active filters, the prototype can be expanded to handle all three-filter types as well as the database file.

In order to implement the active filter prototype, the system first needs to be appropriately configured. This can be done by referring to the Data flow Diagram of the overall system, which will be redrawn for the active filter case. Figure 4.12 in Chapter 4, reproduced here as Fig 5.1, showed the processes associated with the filter type and technology. In relation to this diagram, the processes shown in grey relate to active filters while the non-grey processes

[†] See Section 3.2 for full details.

refer to digital filters. The decomposition of other processes associated with the active filter is shown in Appendix A2.

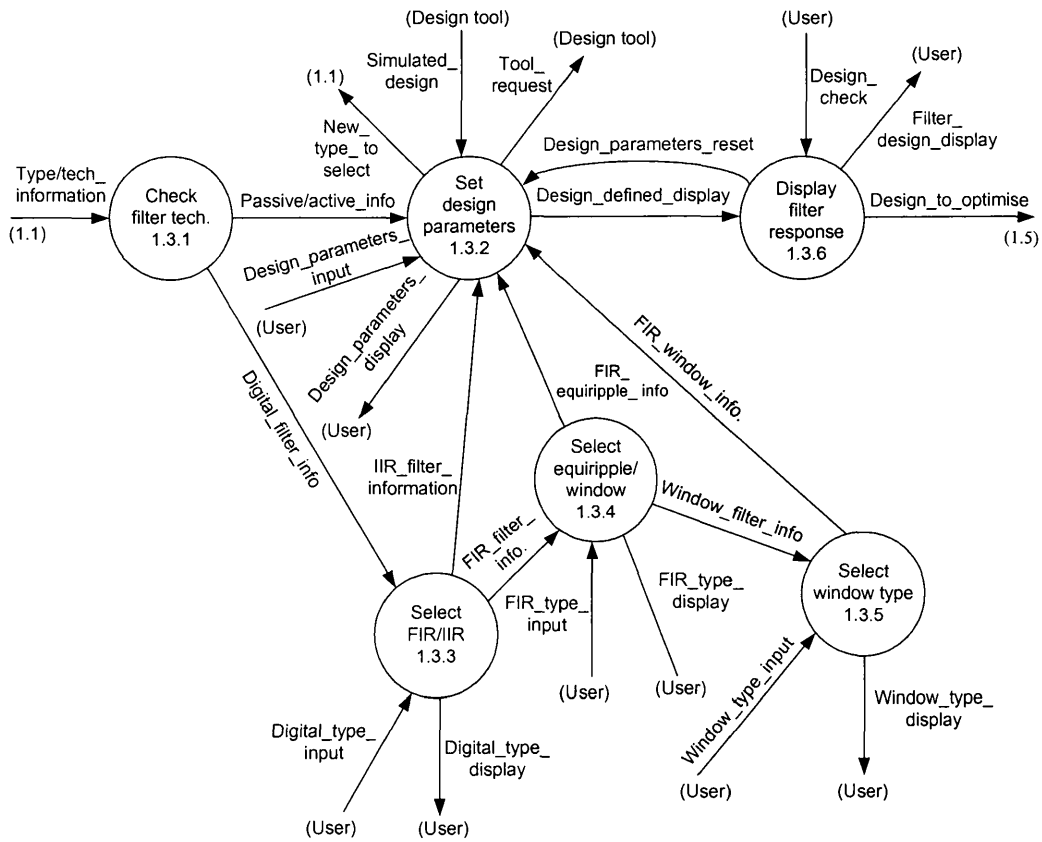


Figure 5.1: *The decomposition of process 1.3 for active filter prototype*

5.3 Database Design with Visual Basic

When building the database application for the active filter demonstrator, it is necessary to put as much care into designing the database as into designing the system software. For instance, the filter database table, records and queries all involve working with objects, and the way these objects are designed will have as great an impact on the application as would any control logic.

Once the database design is completed, attention can be turned to the software and the use of Visual Basic to build the system front end. This front end consists of the forms and code that will be used to manipulate the database. The design of the general database file was discussed in Chapter 3 and 4 and comprises a single table containing the individual filter design records. Because a single table is used, the database structure is kept simple, both in structure and in the ability to establish a link to the Visual Basic User front end. The table,

which is developed from the table of Fig. 3.6 in Chapter 3, now needs to be revised and organised to fit the design case of active filters. The table of active filter design cases is then as shown in Fig. 5.2

cases : Table

Designer	Type	Response	Cutoff Fre	lower cutof	upper cuto	passband ripp	stopband atter	Order	Application
Schematica Software	Lowpass	Butterworth	22				29	6	Noise elimination(Dolby)
Schematica Software	Lowpass	Elliptic	50000			0.1	35	7	Antialiasing
Schematica Software	Highpass	Elliptic	400			0.9	33	4	1/f noise elimination
Schematica Software	Highpass	Chebyshev2	500				30	7	dc blocking
Schematica Software	Bandpass	Elliptic		50000	60000	0.05	25	6	frequency tuning
Schematica Software	Bandstop	Elliptic		49	51	0.1	30	6	reject 50Hz power line noise
Dailey	Lowpass	Chebyshev1	1200			1	32	2	Noise elimination(Dolby)
Linear technology	Lowpass	Butterworth	11				30	4	Antialiasing
Linear technology	Lowpass	Elliptic	20000			0.1	30	4	phased locked loop
Schematica Software	Lowpass	Chebyshev2	100				35	6	Smoothing
Dailey	Highpass	Butterworth	500				40	2	1/f noise elimination
C.Main	Highpass	Chebyshev2	1000				40	5	dc blocking
Linear technology	Highpass	Chebyshev1	30000			0.05	30	8	noise elimination at low frequenc
linear Technology	Highpass	Elliptic	11			0.1	23	8	highpass
S.Reynolds	Bandpass	Butterworth		2000	2400		40	4	graphic equiliser
Linear technology	Bandpass	Elliptic		96000	104000	0.2	25	8	frequency tuning
William	Bandstop	Chebyshev1		100	400	0.5	41	5	bandstop to prevent signal leaka
C.Main	Bandstop	Butterworth		47.5	52.5		44	2	reject 50Hz power line noise

(a) *Table part (I)*

cases : Table

Implementatio	Volume	Component Co	Complexity	Chip features	Operating pow	Cost	Opertating Tem	Notes	Reference
Discrete compor	high	13	medium	general usage	+/-3.5 to +/-18V	low	0 - 70 degree	Cascaded of 2nd o	Schematica Software
Discrete compor	high	30	high	general usage	+/-3.5 to +/-18V	medium	0 - 70 degree	Cascaded of 2nd a	Schematica Software
Discrete compor	high	19	low	general usage	+/-3.5 to +/-18V	low	0 - 70 degree	Cascaded two 2nd	Schematica Software
Discrete compor	low	30	high	general usage	+/-3.5 to +/-18V	high	0 - 70 degree	Cascaded of 2nd a	Schematica Software
Discrete compor	medium	28	medium	general usage	+/-3.5 to +/-18V	medium	0 - 70 degree	Cascaded of 2nd o	Schematica Software
Discrete compor	low	27	medium	general usage	+/-3.5 to +/-18V	high	0 - 70 degree	Cascaded of 2nd o	Schematica Software
Discrete compor	low	7	low	precision	+/-3.5 to +/-18V	low	0-70 degree	2nd order Sallen-K	text book: operational
Discrete compor	low	9	low	extremely easy	3 to +/-5V	low	-40 to 85 degree	Cascadable to forr	www.linear-tech.com
Discrete compor	medium	16	medium	very low noise	-5V to +5V	low	-40 to 85 degree	low distortion	www.linear-tech.com
Discrete compor	high	28	medium	general usage	+/-3.5V to +/-18\	low	0-70 degree	cascaded of three s	Schematica Software
Discrete compor	low	5	low	precision	+/-3.5V to +/-18\	low	0-70 degree	Sallen-Key	text book: operational
discrete compor	high	12	low	general usage	+/-3.5V to +/-18\	low	0-70 degree	higher order can be	Dr. C.Main
discrete compor	medium	15	high	very low noise	-5V to +5V	high	-40 to 85 degree	low distortion	www.linear-tech.com
discrete compor	low	18	high	very low noise	-5V to +5V	high	-40 to 85 degree	low distortion	www.linear-tech.com
discrete compor	medium	14	low	general usage	+/-3.5 to +/-18V	low	0 - 70 degree	obtained by cascac	text book: operational
discrete compor	low	19	high	very low noise	-5V to +5V	high	-40 to 85 degree	low distortion	www.linear-tech.com
discrete compor	high	28	low	precision	+/-3.5V to +/-18\	low	0- 70 degree	cheap as it uses q	text book: electronic fi
discrete compor	low	9	low	precision	+/-3.5V to +/-18\	low	0-70 degree	single null used for	text book: electronic fi

(b) *Table part (II)*

Figure 5.2: *The database table for active filter prototype*

The background to this table was discussed in Chapters 2 and 3; the meaning of the individual fields of Fig. 5.2 was also set out in Table 3.2 in Chapter 3

5.3.1 Design Criteria Fields

The fields in the filter design database table such as '*Cost*', '*Component count*', '*Volume*' and '*Complexity*' are used by the user to set design preferences which can then be used when searching for a design match. As discussed in Chapter 3, crisp values are used at this stage in the implementation.

The '*Cost*' field of each record is calculated from the final average of the cost of each element in the filter circuit. This will result in a cost value for each design, which was then assigned as being either a low, medium or high cost solution. For the purpose of the demonstrator the (crisp) boundaries were assigned as follows:

- Cost $\leq$£6 ⇨ Low cost
- £6 < Cost < £10 ⇨ Medium cost
- Cost $\geq$£10 ⇨ High cost

Volume refers to anticipate production volumes and can be assigned as either a low volume design, a medium volume design or a high volume design. In calculating the cost an inverse relationship has been assumed in this instance between cost and volume, though it would be up to individual companies to specify the relationship appropriate to themselves.

5.4 Data Access

A common means of controlling the interaction with a database is to use the Data Access Objects (DAO) approach. Using DAO, it would be possible to write an explicit code for connecting to a database, defining Recordsets[†], manipulating field values and so forth. Embedded Structured Query Language (SQL) commands can be used effectively to retrieve data from and update the database. Using SQL to define the Recordsets and manipulating the rows individually using DAO commands is the procedure that will be used here.

Linking an Access database to Visual Basic using the data control constructs is done by first creating a Visual Basic form containing text boxes, combo boxes and labels representing the data entries or fields in the database table. Data control, frame and command buttons are also placed on the form as shown in Fig. 5.3.

[†] A recordset is defined as a set of records held in memory

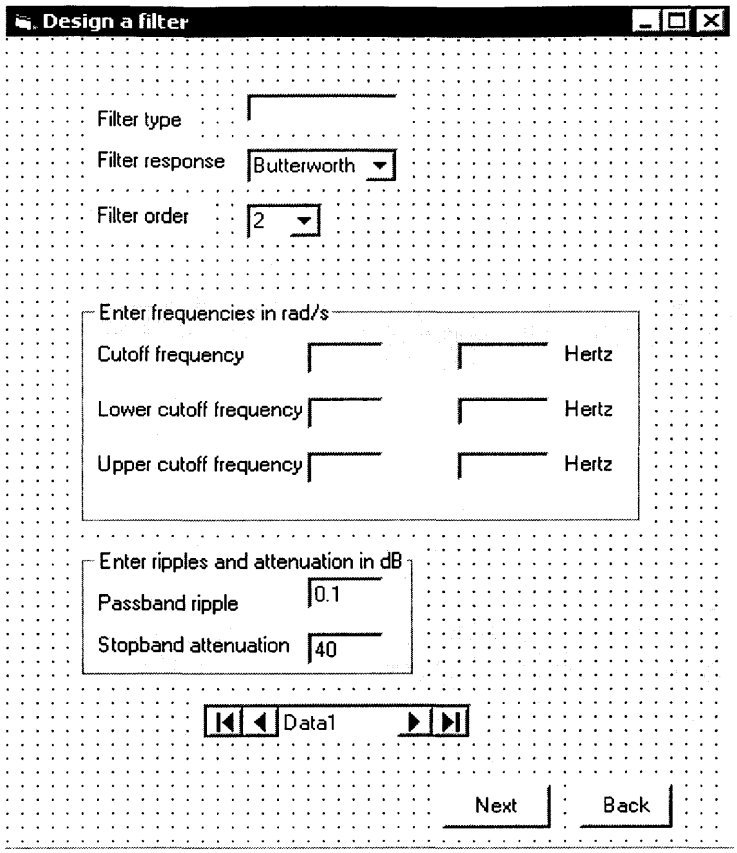


Figure 5.3: *Visual Basic controls*

Some of the properties of the Data Control '*Data1*' are shown in Fig. 5.4. The link to the database filter table uses the significant properties *Databasename*, which is the file path that links the data control to the database, and *Recordsource* that can be seen by scrolling down the properties window which is set to the name of the source of the data. In this case the filter design database table.

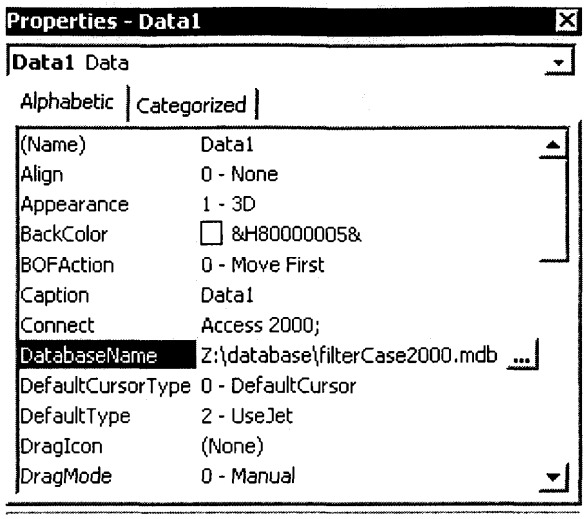


Figure 5.4: *Visual Basic properties of Data Control 'Data1'*

Linking a data control to a database creates a Recordset. The data control will point to just one of these records at any one time and allows scrolling through the records in the Recordset. The selected record will then be displayed in the frame boxes. To tie any particular text box to the Recordset, two properties are to be set for that text box. The property *Datasource* has to be set to the name of data control and the *Datafield* property has to be set to the required field on the database table to be displayed, for example '*Response*'. The other important control is the *FlexGrid* control, this provides a datasheet-type display of a table's records, displaying all the records in a table as separate rows with columns for each field and scroll bars if necessary (see Fig 5.5).

The user can use the arrow buttons in Fig. 5.5 for navigating through the records of the database. In Visual Basic there are a number of methods can be used, which can be invoked in code; these methods belong to the *Recordset* property object of the data control. Some of the commonly used methods are shown in Table 5.1.

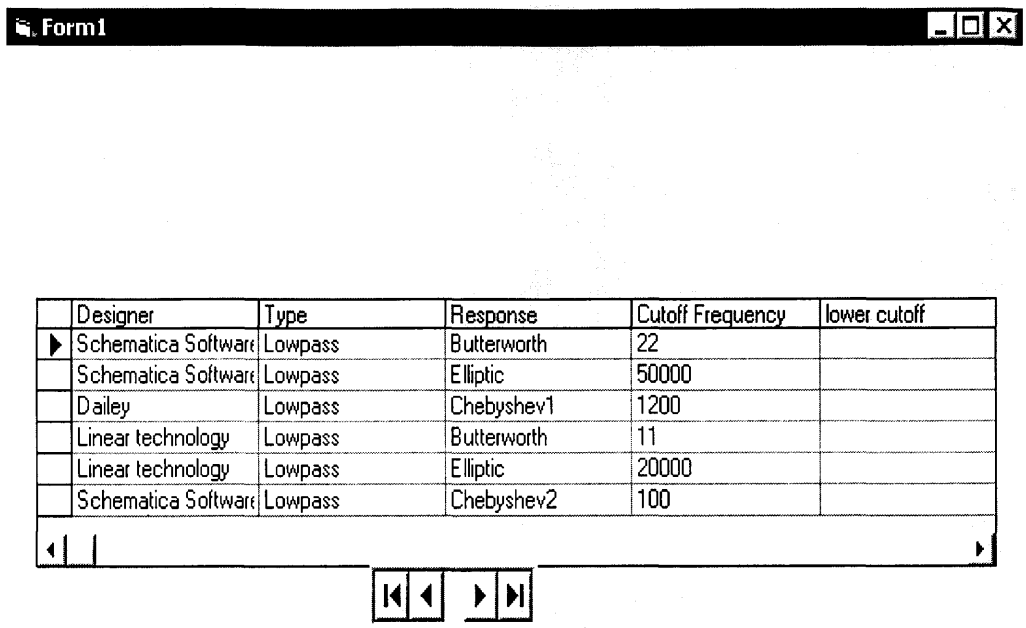


Figure 5.5: *FlexGrid control*

Table 5.1: *Recordset access methods*

MoveFirst	Move to the first record
MoveLast	Move to the last record
MoveNext	Move to the next record
MovePrevious	Move to previous record
Move [+/-]n	Move specified number of records from Current Record (use *AbsolutePosition* and *PercentPosition* properties to move through the Recordset)
AddNew	Creates a new blank record whose field values can now be set (requires the database and Recordset to allow updates)
Update	Applies any update (field edits or new record) to the Recordset
Delete	Deletes the current record
Close	Closes down a Recordset

The methods in the above table, along with other processing functions not mentioned above, can be used for navigating through the records or to find a specific record within the Recordset. In order to retrieve, display and update data from database table efficiently and more flexible, the Structured Query Language (SQL) along with the methods in Table 5.1 are used.

5.4.1 Structured Query Language (SQL)

Queries allow the programmer to retrieve data from the database and to update it. In most databases, including Access, SQL*Server, ORACLE, Sybase, INGRES and others, SQL is the main method used for this. Indeed, SQL is the primary language – and may almost be considered as the de facto standard - for communicating with a relational database management system (RDBMS). SQL can be used interactively; typing an SQL statement directly into Access and getting the result immediately or it can be embedded in a Visual Basic programme, which is the method used here.

The interactive use of SQL is an effective way to develop SQL statements that are later going to become embedded in a Visual Basic program as it provides an effective means of test and validation.

However, the output from an interactive SQL query is in a primitive form, just columns of data. Users may want to perform a query as part of a more extensive process involving other queries, programs, inputs and outputs. The Visual Basic code will tie all these processes together and relate them logically within a structured user interface.

SQL SELECT statement:

The purpose of the SQL SELECT statement is to retrieve and display data gathered from one or more database tables. SELECT is the most frequently used SQL command and can be used interactively to obtain immediate answers to queries from the command line or embedded in a host program. The basic syntax of the SELECT statement for a single table is shown in Fig. 5.6.

SELECT	col_name,...
FROM	table_name
WHERE	condition
AND/OR	condition
GROUP BY	col_name
HAVING	condition
ORDER BY	col,

Figure 5.6: *SELECT syntax for a single table*

The sequence of processing in a SELECT command is:

SELECT	Specifies which results will be output by specifying fields or use '*' for all fields.
FROM	Specifies the table to be accessed.
WHERE	Filters the rows on some condition.
GROUP BY	Forms a single row from a group of rows.
HAVING	Filters the groups on some condition.
ORDER BY	Determines the order of the output rows.

The following example illustrates the use of SELECT command to retrieve antialiasing filter design cases from the database table '*Cases*'. For simplicity, interactively the command line in Microsoft Access is used as follows:

```
SELECT * FROM Cases
WHERE Application = 'Antialiasing';
```

Reference to Fig. 5.2 shows that there are two antialiasing applications identified and the output from the above query is therefore as shown in Fig. 5.7.

	Designer	Type	Response	Cutoff Frequency	lower cutoff	upper cutoff
▶	Schematica Softwar	Lowpass	Elliptic	50000		
	Linear technology	Lowpass	Butterworth	11		
*						

Figure 5.7: *The output of filter design query for antialiasing filter applications*

The WHERE clause, defines a condition which must be met before it is included in the returned table. It may use relational operators such as (<, <=, =, >, >=, <>) or the LIKE operator to compare column and/or constant values. This comparison may be combined using the Boolean operators (AND, OR, NOT). As an example, suppose that a Butterworth lowpass filter with cut-off frequency >= 10 is required, then the following query may be used:

```
SELECT * FROM Cases
WHERE filter_type = 'Lowpass' and filter_response
      = 'Butterworth' and cutoff_frequency >= 10;
```

The output from this query is shown in Fig. 5.8

	Designer	Type	Response	Cutoff_Fre	lower cutol	upper cuto	passband ripp	stopband atter	Order	Application
▶	Schematica Software	Lowpass	Butterworth	22				29	6	Noise elimination(Dolby)
	Linear technology	Lowpass	Butterworth	11				30	4	Antialiasing

Figure 5.8: *The output of filter design query of a Butterworth lowpass filter of frequency ≥ 10Hz*

In this project, SQL statements are embedded in the VB application code to enable the user to retrieve the required data through the system interface. The full range of embedded SQL statements as part of the overall code can be seen in Appendix A.

5.5 Coding

In this section, the physical design specifications of the system established in Chapter 4 are turned into working code. Once coding has begun, the testing process can also begin and proceed in parallel with a program module not usually being considered as finished until the tests for that program module have been passed. Thus programming and testing are tightly coupled.

The coding and testing for the expert system described in this thesis is done by coding and testing each module individually. The modules are then integrated and tested as a part of a larger block of code, and then as a part of a larger system. The coding of the individual modules typically takes place either in a top-down order referred to the system diagram or by following the hierarchical reference numbers of each module from the Data Flow Diagram.

The individual module code will thus be a translation of the processes in the Data Flow Diagram to produce the Visual Basic code. Referring to Fig. 5.9[†], the coding will start at the first module in the dialogue diagram, item '0', the 'Splash form'. The relevant code is given in section 4 of Appendix A.

[†] See Section 4.5.2 for details

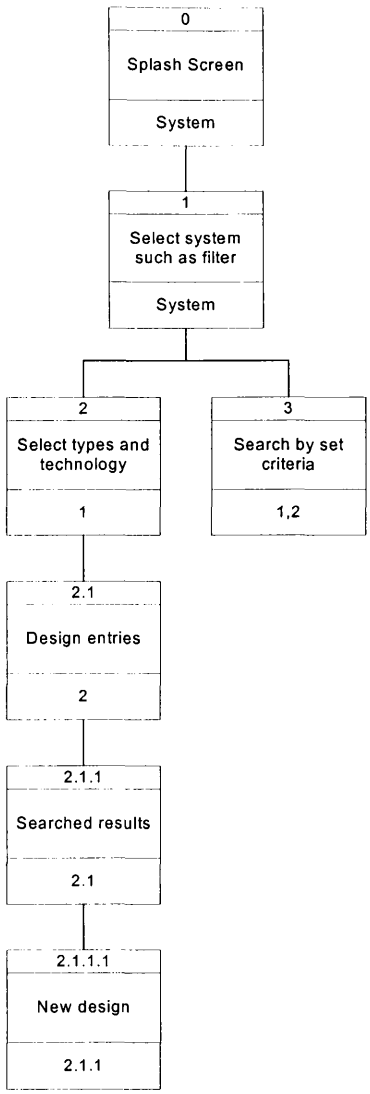


Figure 5.9: *Dialogue diagram for filter design expert system*

5.5.1 'Splash form' (Item 0)

Almost every off-the-shelf or professional application displays a '*Splash Screen*' when it launches. This practice came about largely as way to keep the user happy while the programme was initialising. Splash screens are so ubiquitous in windows that a programme is now expected to have one. The Splash Screen used for the electronic design expert system is shown in Fig. 5.10.

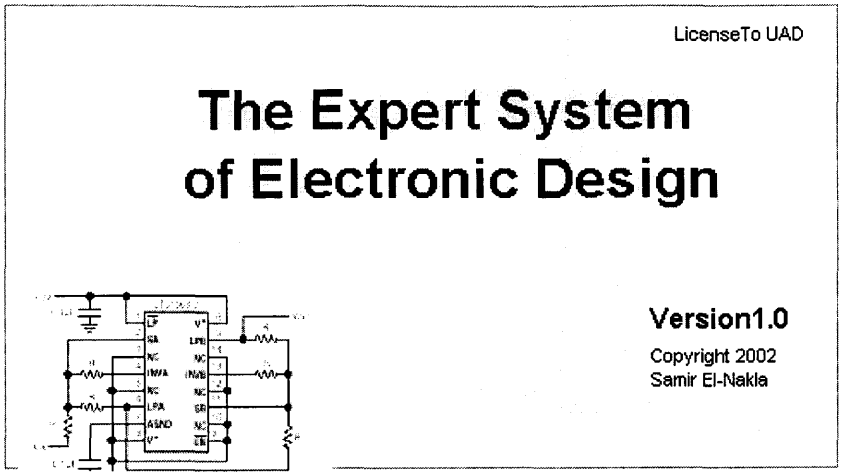


Figure 5.10: *Splash Screen for the filter design expert system*

5.5.2 'Select the system' (Item 1)

This form comes straight after the splash screen and is where the user can select the required electronic system domain. The combo boxes in the form provide a list of the five most common electronic systems; amplifier, digital signal processor (DSP), filter, microprocessor and microcomputer (see Fig. 5.11).

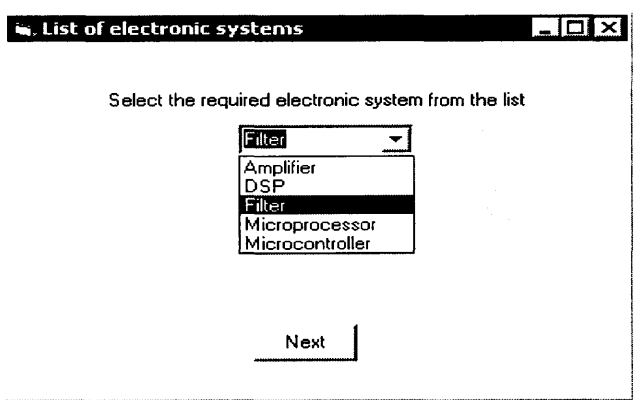


Figure 5.11: *Item '1' form where the user can select a system*

As a filter is the exemplar used in this research, the filter item in the combo box list will be set as the default and whenever a user tries to select any electronic system other than this a prompt message appears to say that only the filter option can be selected. An example of such a message is shown in Fig. 5.12.

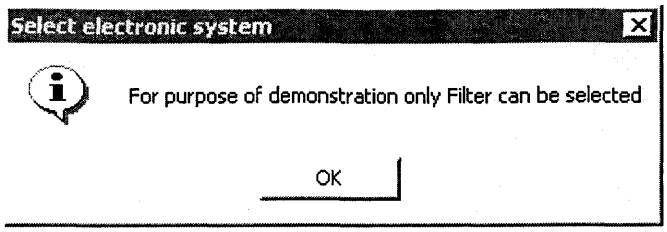


Figure 5.12: *A message box prompting the user to select filter*

Another form is that from which the user can opt to go down either the design by type route or the design by application route. This form is included to improve usability and is opened when the user clicks the '*Next*' command button on the form of Fig. 5.11. When the new form opens (see Fig. 5.13) it asks the user to select one of two design options. It also has two command buttons, '*Back*' takes the user back to the previous form and the '*Next*' button, which takes the user to the next form.

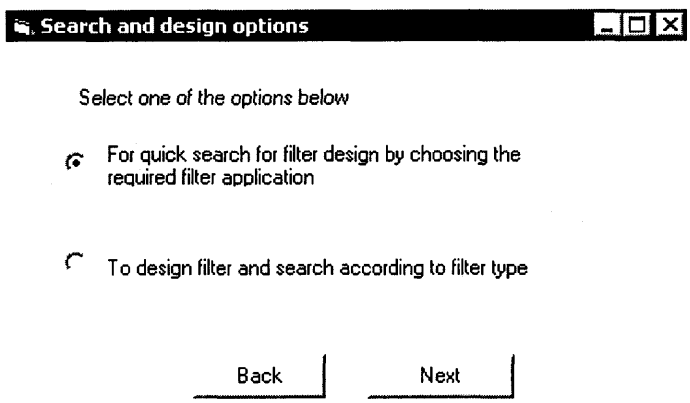


Figure 5.13: *Choice of design options*

5.5.3 'Select filter type and technology' (Item 2)

The user reaches this form, which will enable the user to select the filter type and the filter technology (see Fig. 5.14), by selecting the '*Design by Type*' option on the form of Fig. 5.11. Filter type may be lowpass, highpass, bandpass or bandstop, with lowpass set as default. Filter technologies are passive, active or digital, with active chosen for the demonstrator. If any of the other filter technologies is chosen a message box appears says to the user that for purpose of demonstration only an active filter can be selected (see Fig 5.15). The '*Back*' command button of Fig. 5.14 takes the user to the previous form.

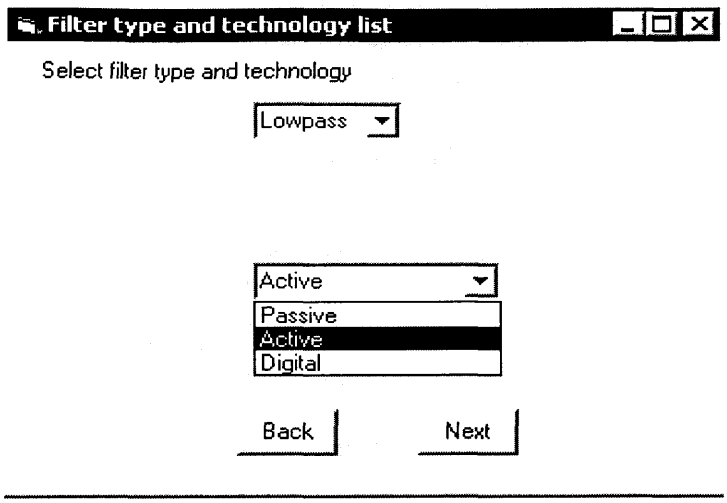


Figure 5.14: *Filter type and technology form*

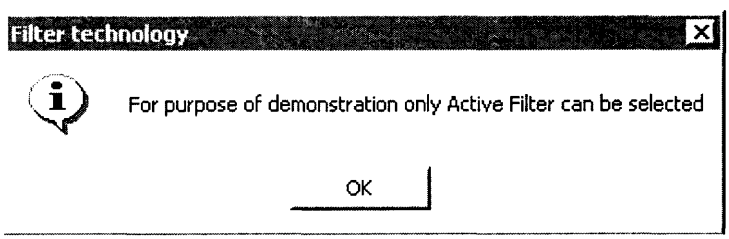


Figure 5.15: *Message box prompting the user to select an active filter*

5.5.4 'Design entries' (Item 2.1)

This is where the user enters (or selects) the filter specification. Initially, the form has default values as shown in Fig. 5.16, which can be modified by the user as required. The '*Frequencies*' frame will vary with the selected filter type. Thus for the default lowpass filter the user is asked to define the cut-off frequency

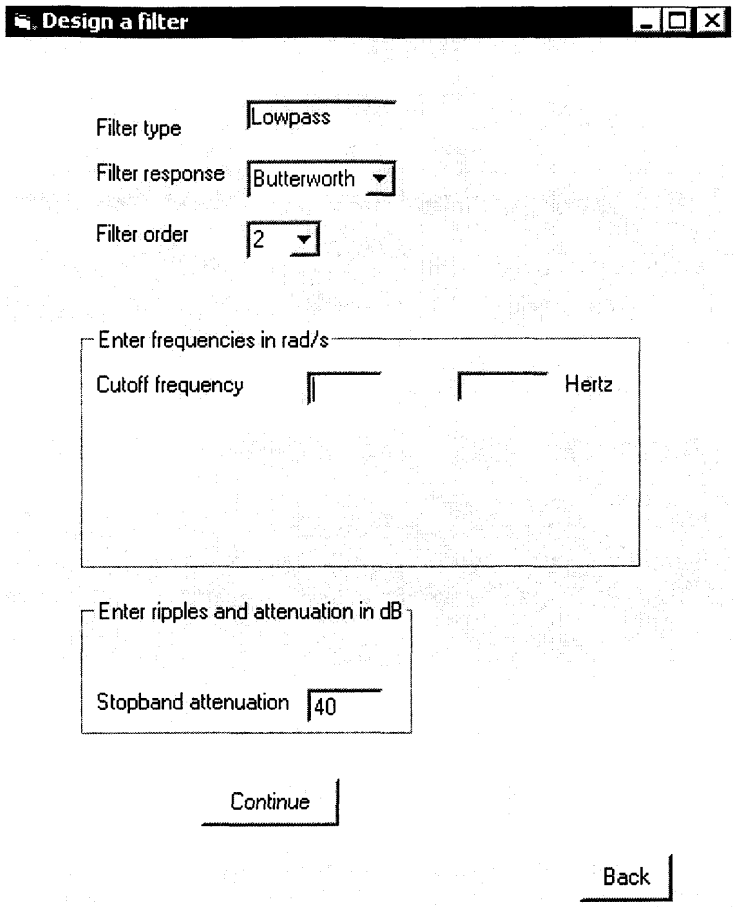


Figure 5.16: *A filter design entries form with default values*

However, if the user selects, say, a bandpass filter then the requirement will change to that shown in Fig. 5.17 where both the upper and lower cut-off frequencies are required to be entered. The frequency unit entered is in rad/s, which is compatible with the Matlab library, but the user sees the equivalent in Hertz. Also seen in Fig 5.17 is an additional box where the user is asked to enter the requirement for passband ripple and stopband attenuation. This box will appear when certain filter types, in this case Elliptic, are selected and have values in dB. In the case of a novice user, help and advice can be obtained for each of these screen features; see section 5.5.7 for discussion of embedded help.

If a user does not enter a value in a box with no default value a message box appears asking the user to provide the information. Also, should a user, for example, enter a value for a lower cut-off frequency greater than the upper cut-off frequency a message appears to warn that the lower cut-off frequency must be less than the upper cut-off frequency.

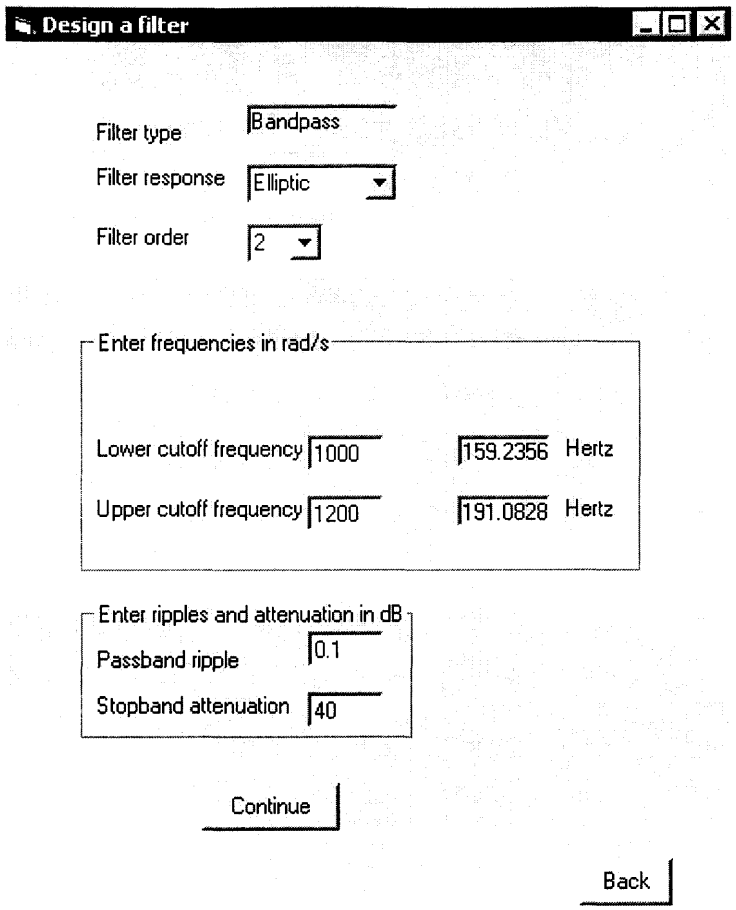


Figure 5.17: *Filter design entry form with specific values*

For the demonstrator, in order to bound the design and the user choice, the frequency range of filter design has to be between 1 and 100,000 rad/s inclusive. If the user enters a value outside that range then the message of Fig 5.18 appears.

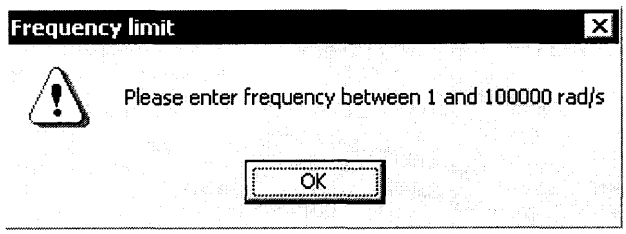


Figure 5.18: *Frequency limit message*

Similarly, the filter passband ripple is set to have a range between 0.01 and 3 dB and the filter stopband attenuation a value between 1 and 100 dB. Again, if the user enters a value outside these ranges, an error message will be displayed. If all entered values are consistent

and within required limits two other command buttons are enabled. One of these launches the Matlab application to generate the frequency and phase responses for the entered design values. The other is the '*Search*' command, which initiates the search of the filter design database and displays the results.

5.5.5 'Searched results' (Item 2.1.1)

Once the user is satisfied with the design specification they can set additional criteria for a search. Figure 5.19 shows these criteria with their default values.

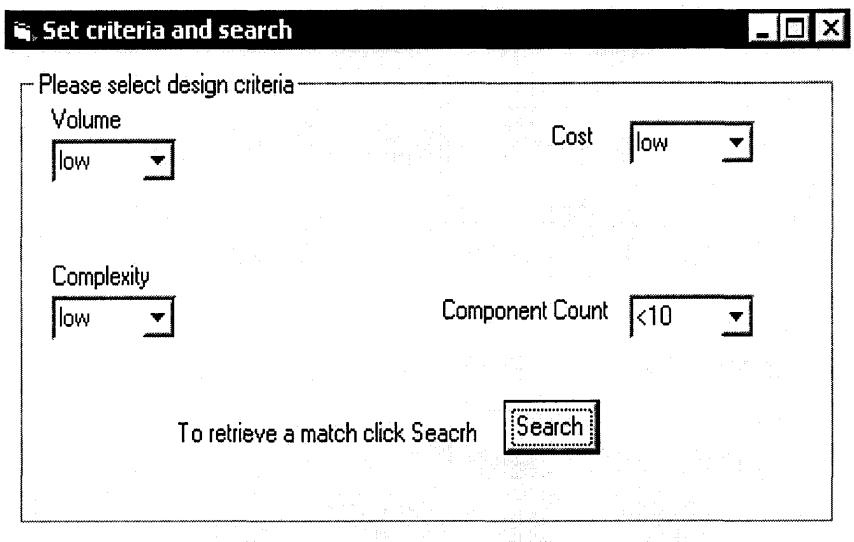


Figure 5.19: *Search criteria with default values*

Once the search criteria have been set, the '*Search*' command initiates the procedure. The output from the search (see Fig. 5.20) will be to display a list of design matches, each with a percentage score indicating the degree of match. The higher percentage the closer the match to the user requirements.

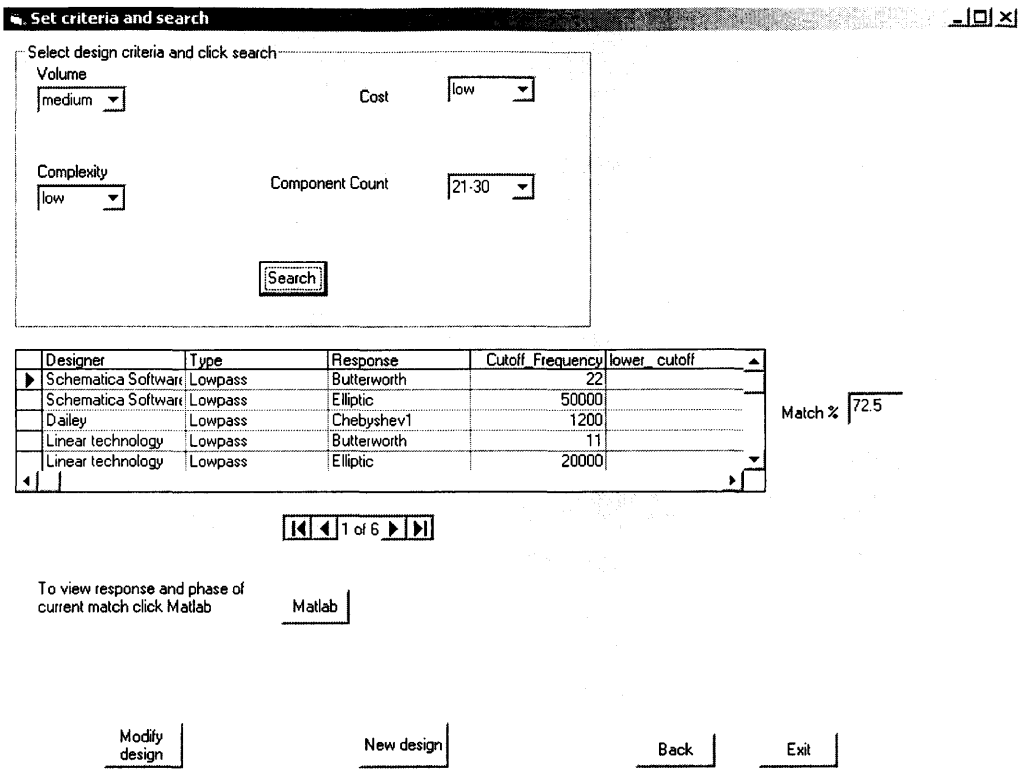


Figure 5.20: *Search results*

The user can view all the returned records using the arrows of the data control. They can also scroll horizontally to view all the fields in a record. If the user makes a choice of a specific solution, then clicking on that record makes it the current one. The user then has the option of viewing the frequency response and phase of the selected design using the Matlab command button.

The percentage value on the form of Fig. 5.20 represents the returned match based on the user-defined criteria. The user selected design criteria are compared to those in the database and a score given accordingly. These scores will typically be in the range from '1', indicating little or no match to to'10' representing a full match. For instance, the scores for design volume are distributed as shown in Table 5.2. The volume in the vertical column represents the user-desired volume where as the volume on the horizontal represents the information in the database.

Table 5.2: *Scores for filter design volume*

Desired Volume ↓	Volume in database file →		
Volume	*Low*	*Medium*	*High*
Low	10	5	2
Medium	5	10	5
High	2	5	10

The same approach can be applied to design criteria such as cost, complexity and component count. The corresponding scores are being shown in Tables 5.3, 5.4 and 5.5 respectively.

Table 5.3: *Scores for filter design cost*

Desired Cost ↓	Cost in database file →		
Cost	*Low*	*Medium*	*High*
Low	10	6	2
Medium	4	10	6
High	3	6	10

Table 5.4: *Scores for filter design complexity*

Desired Complexity ↓	Complexity in database file →		
Complexity	*Low*	*Medium*	*High*
Low	10	5	2
Medium	6	10	4
High	2	6	10

Table 5.5: *Scores for filter design component count*

Desired Component count ↓	Component count in database file →			
Component count	*<10*	*10 – 20*	*20 – 30*	*>30*
<10	10	7	4	1
10 – 20	8	10	8	3
20 – 30	4	8	10	6
>30	2	6	8	10

The percentage match as displayed on Fig. 5.20 is calculated for each record by finding the total score over all criteria and then presenting this as a percentage of the ideal match. The score for the selected current record is then displayed on the form. This will be demonstrated in a design example at the end of this section.

The user may chose the most appropriate match from those returned by the search. This design can then be modified by selecting the '*Modify design*' command. This brings up the screen of Fig 5.21, which, in this instance, provides a link to the filter design tools such as FILTERCAD and SHECMATICA. Once the required modifications have been made to the selected design, it can be stored back in the database as a new design. At this stage the link to the design tools has not been automated, though this would be the case in a fully working system.

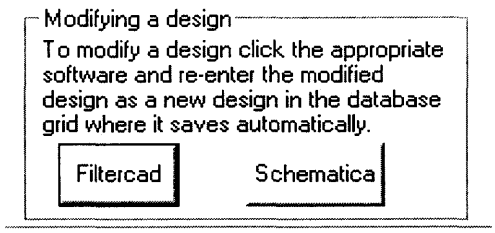


Figure 5.21: *Filter design tool selection*

If the user is not satisfied by any of the filter design records returned by the search, they have the option of constructing a new or custom design. This can be done by selecting the '*New Design*' option on Fig 5.20. The resulting screen (Fig. 5.22) will contain all the filter design fields and defaults from the search but also includes access to the design tools. There is also one other additional field '*Designer*' which is used to identify the source or nature of the new design. Once a design is complete, the user can save it as a new design in the database. This is done by the '*Save and Exit*' command.

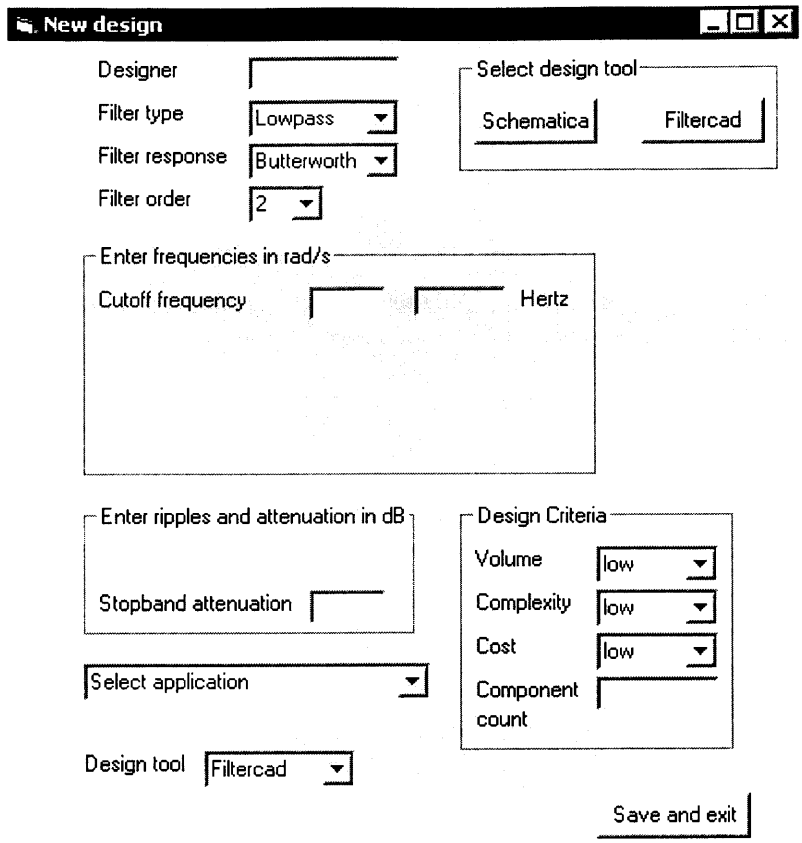


Figure 5.22: *New design entries form*

5.5.6 'Search by filter application' (Item 3)

As an alternative to the approach described in the proceeding sections a user may opt to search by application as an option shown in Fig. 5.23. This is the default option and is referred to as the quick search option.

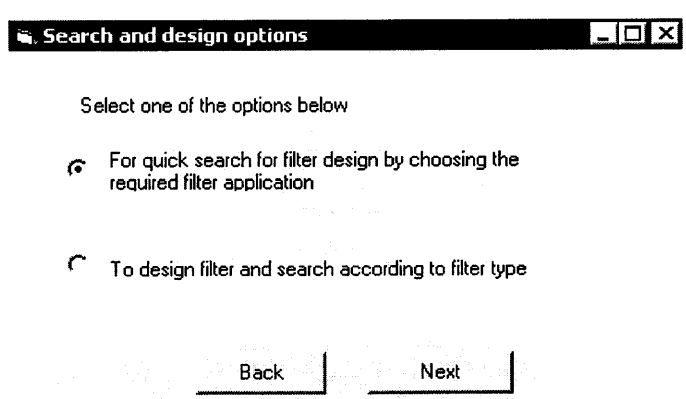


Figure 5.23: *Form giving the user two design options*

On selecting this option, the screen of Fig. 5.24 opens asking the user to select the required filter application.

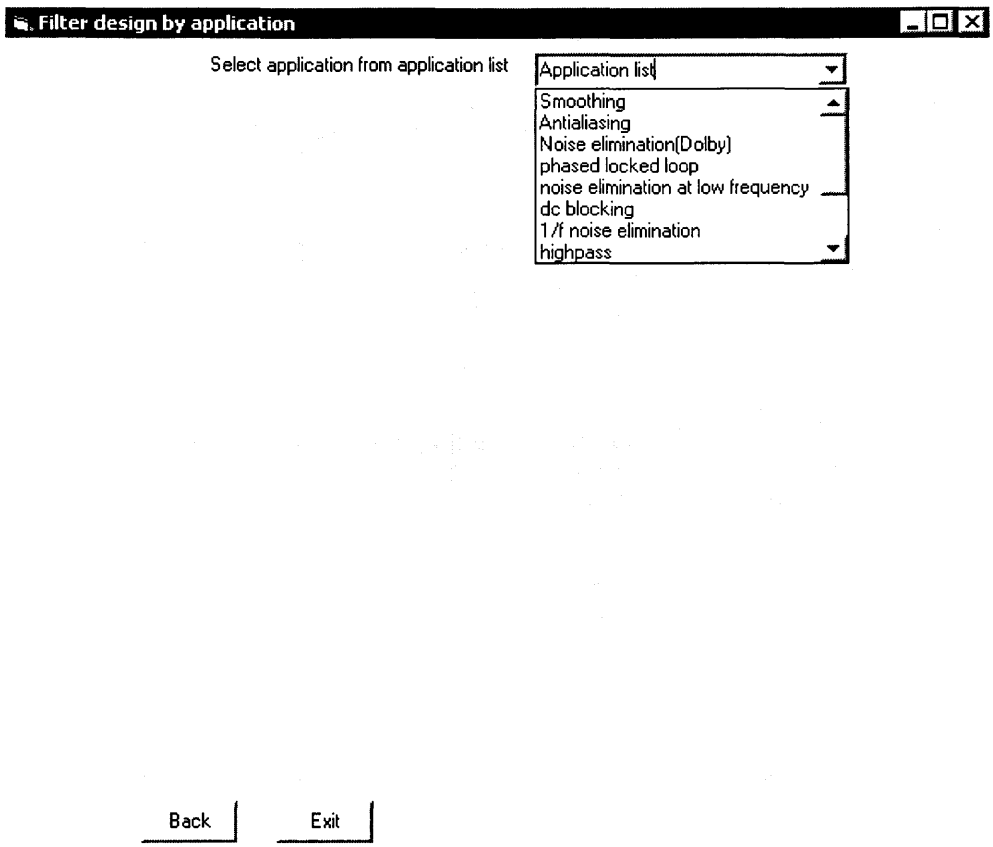


Figure 5.24: *Form with a list of filter applications*

Once the user has selected the desired application, the screen of Fig. 5.25 appears containing the default values for volume, cost, complexity and component count. Following selection of the desired criteria, the command '*Search*' initiates the search of the database to return filter designs matching the specified application together with their match to the criteria.

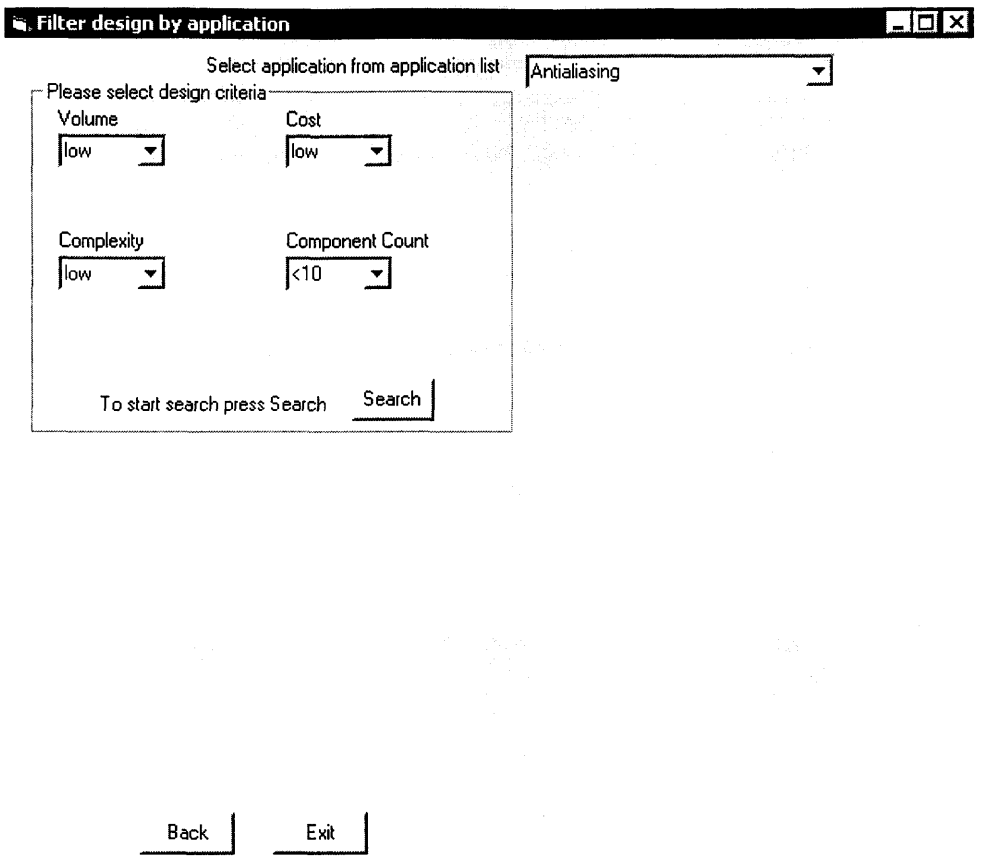


Figure 5.25: *Filter design criteria with a default value*

The algorithm used in calculating the match is exactly the same as that used previously in calculating the match based on filter characteristics and type. The returned information has the form shown in Fig. 5.26. The '*Matlab*' command is used as before to obtain the frequency and phase response of the current record. As before, the user has available a '*Modify Design*' option to provide access to filter design tools.

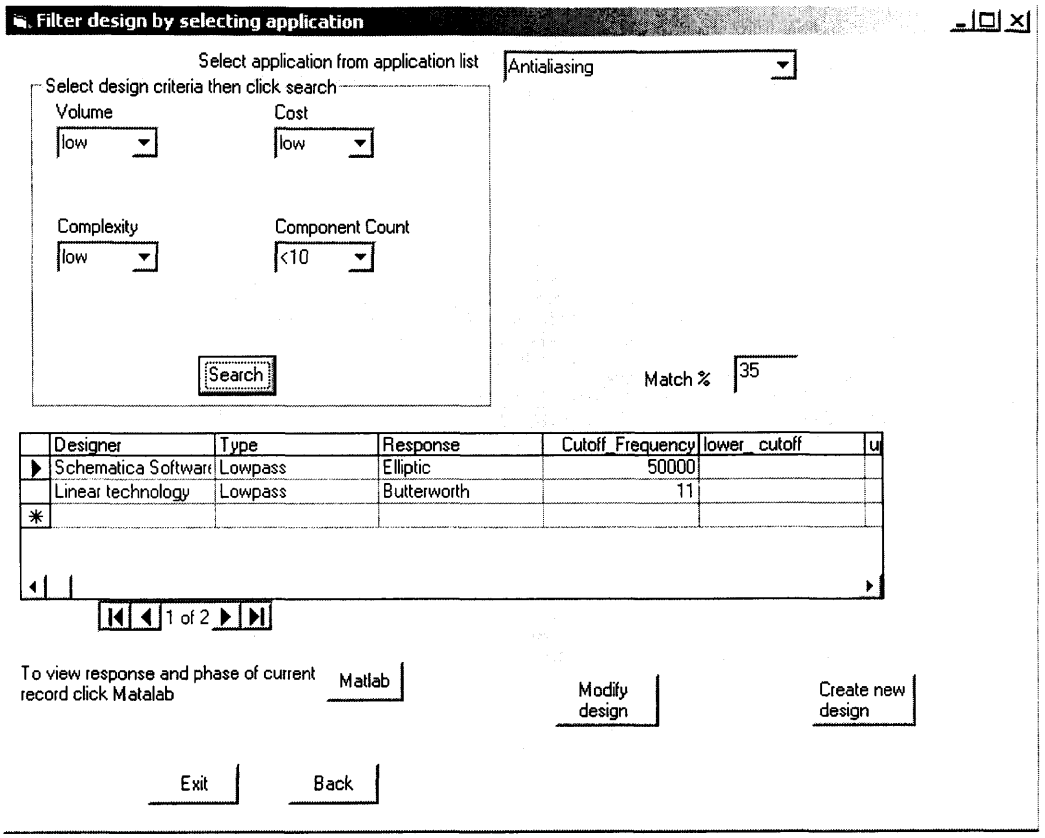


Figure 5.26: *The output of the filter design search*

If the user is not satisfied with any of the returned filters they can use the '*Create new design*' command to take them back to the filter design dialogue structure (see Fig 5.27).

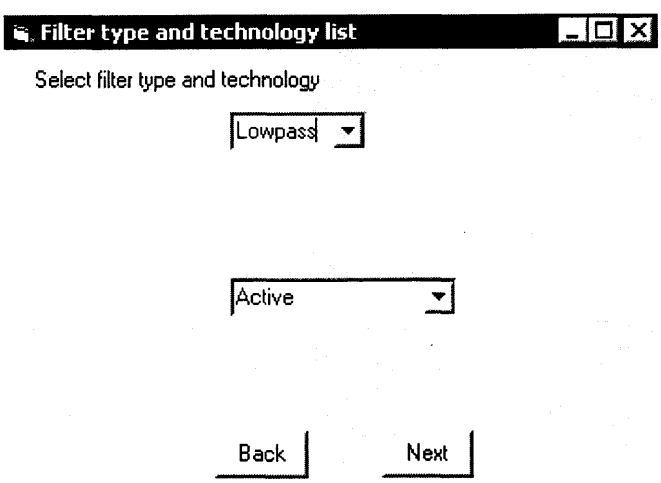


Figure 5.27: *Filter type and technology selection option*

The other options on Fig 5.26 are the '*Back*' command which takes the user to the previous form, while the '*Exit*' command terminates the application subject to confirmation by the user.

5.5.7 System Help

Suppose that the user when entering the filter design variables needs help in understanding filter knowledge and assistance in choosing the filter parameters. The filter design knowledge acquired from human experts and other sources can be represented in a variety of ways[†]. For this demonstrator tool, system level help about the filter response and selection criteria is chosen.

Access to the help feature is by clicking on the '*Help*' command button in Fig. 5.28, identified by a question mark. On clicking this button, a help window is opened as shown in Fig. 5.29. This displays the contents of the help file, including hyperlink titles and subtitles that take the user to the required help window. For instance, if the user needs help about filter response, then by clicking on the "*Response selection criteria*" link, the related help window appears as shown in Fig. 5.30.

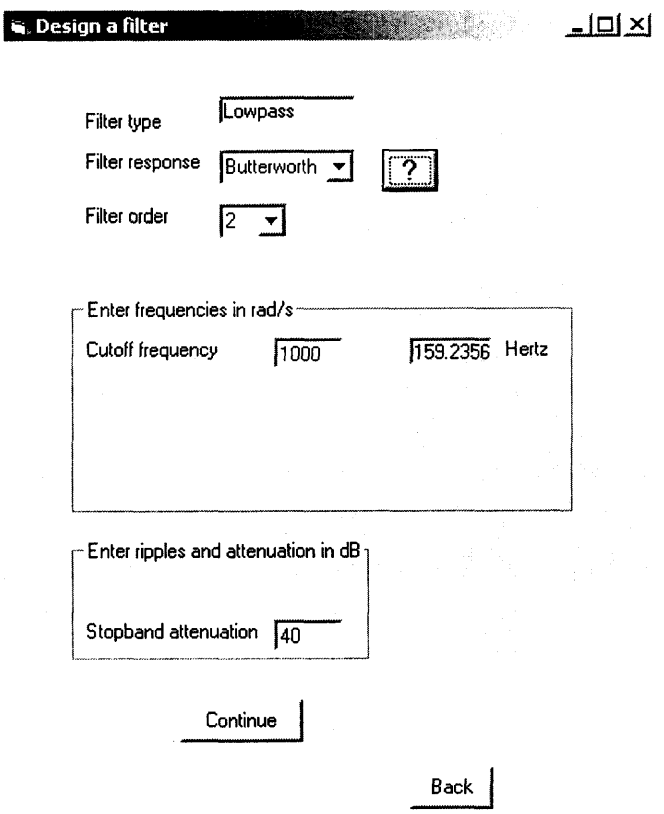


Fig. 5.28: *Filter response command help*

[†] See Section 4.5.1 for different types of help

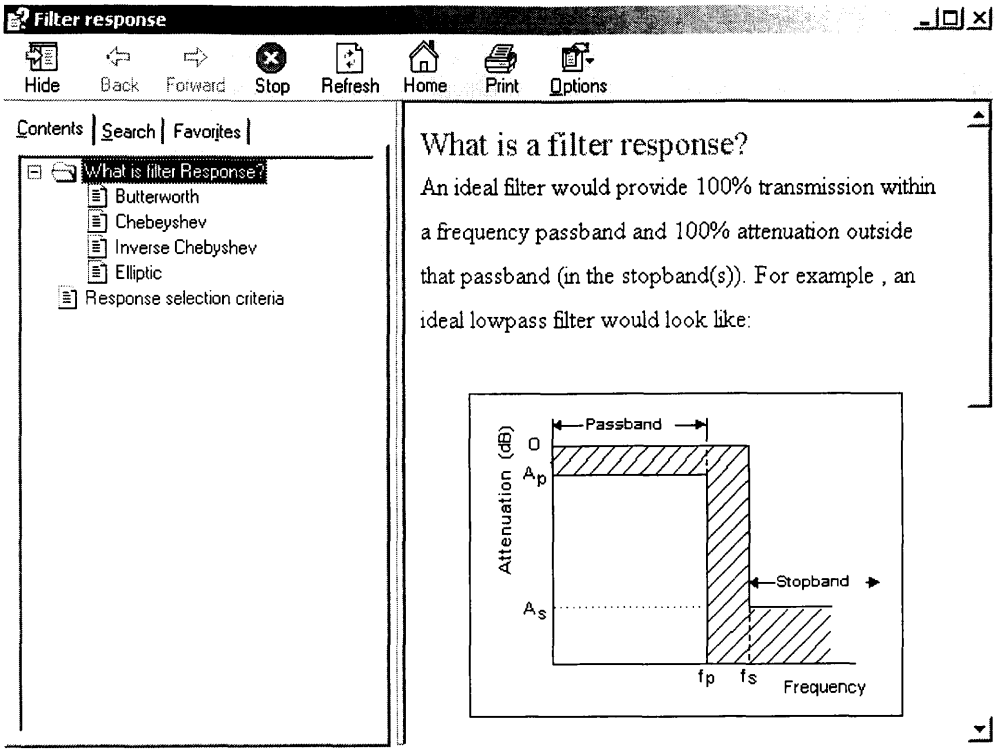


Figure 5.29: *Online help of filter response*

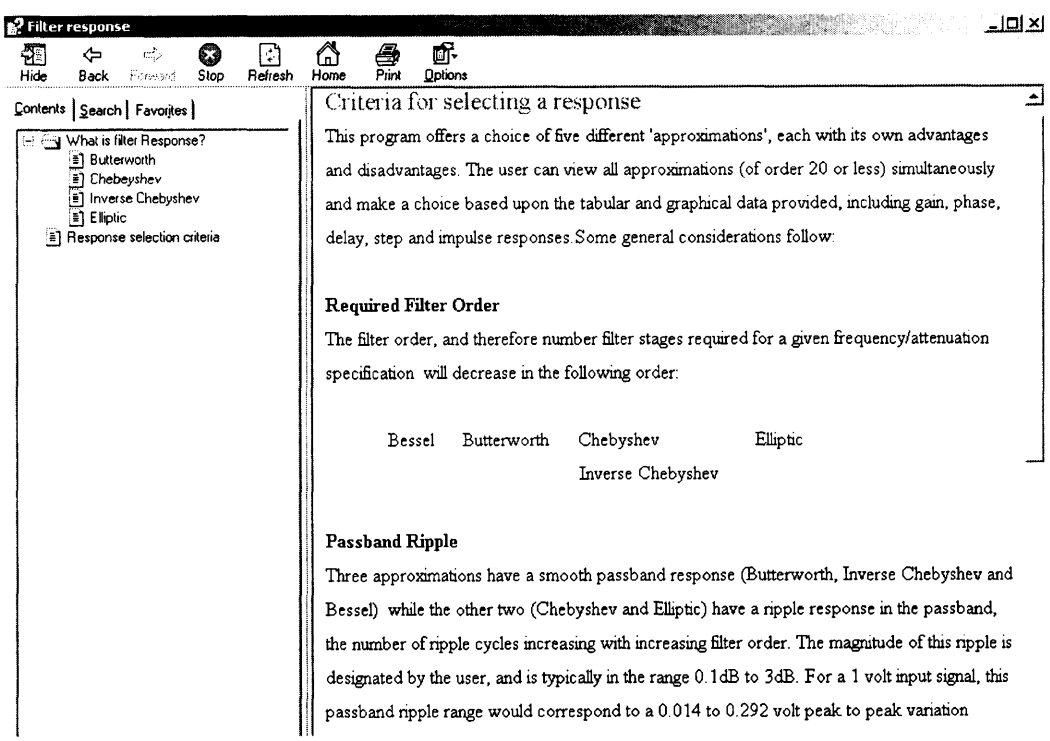


Figure 5.30: *Help window for response selection criteria*

5.5.8 Design Tools

The Matlab software package has been linked to this application only to display the frequency and phase response of any filter design parameters entered or a retrieved design. The availability of mathematical filter design functions as part of the Matlab library made such a link beneficial in relation to time saving rather than having to create the software for the filter functions from scratch. This is achieved by storing the filter design variables in a file and calling it from Matlab, the related code is shown in Appendix A.

The other tools linked to this software application are dedicated filter design tools and are used in either modifying an existing design or in helping to create a new design. Two of the filter design tools were linked to the filter design expert system, FILTERCAD and SCHEMATICA. For instance, on clicking the 'FILTERCAD' command button, the related window is opened as shown in Fig. 5.31. At this stage the user can freely change and simulate the current filter with the aid of these tools. To automate the data transfer to and from the design tools and the system application, system compatibility and advanced code development are needed, therefore it will be including in any future system, the full working version. At this stage the system has been demonstrated manually.

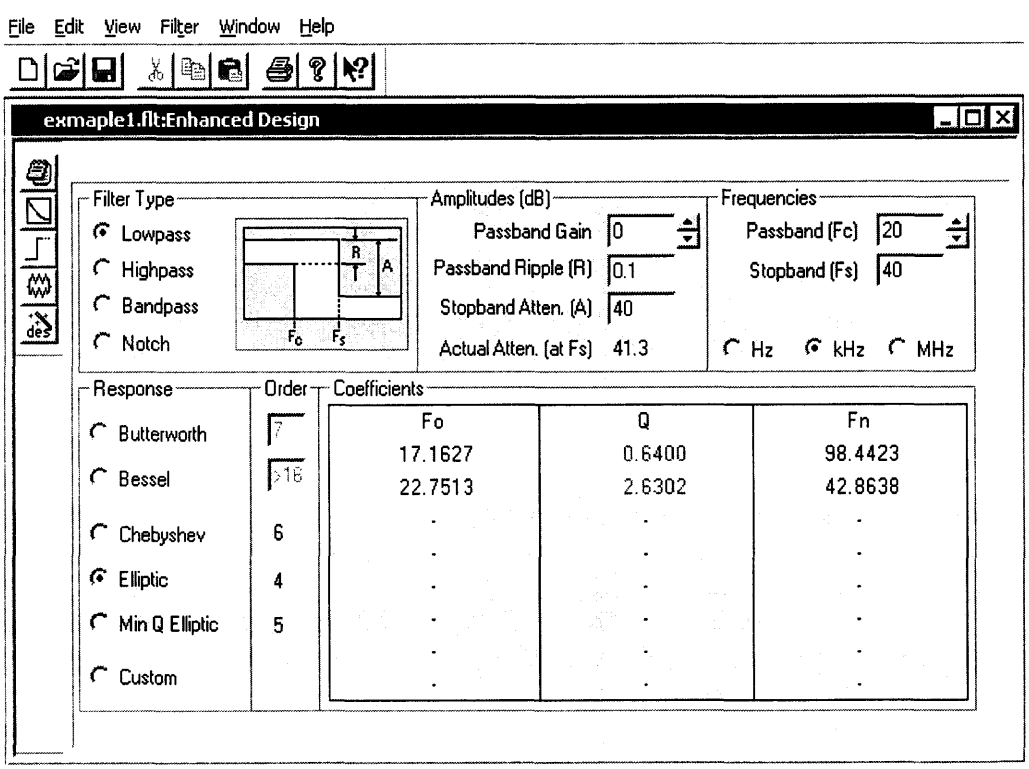


Figure 5.31: *TheFilterCAD design tool*[4]

Design example:

Suppose that a user needs to design a lowpass, forth order, Butterworth filter with a cut-off frequency of 10,000 rad/s and 50 dB stopband attenuation. Suppose also that the user decides to choose the system design option according to filter type not application. Following the entry of the data, the screen of Fig. 5.32 is seen. As a lowpass filter is selected only the cut-off frequency is visible and as a Butterworth response is chosen there is no value required for passband ripple and it is therefore hidden. On selecting the Matlab command, Fig. 5.33 is obtained showing the frequency and phase response of the filter. The user at this stage can view the response and decide on the appropriate cut-off frequency.

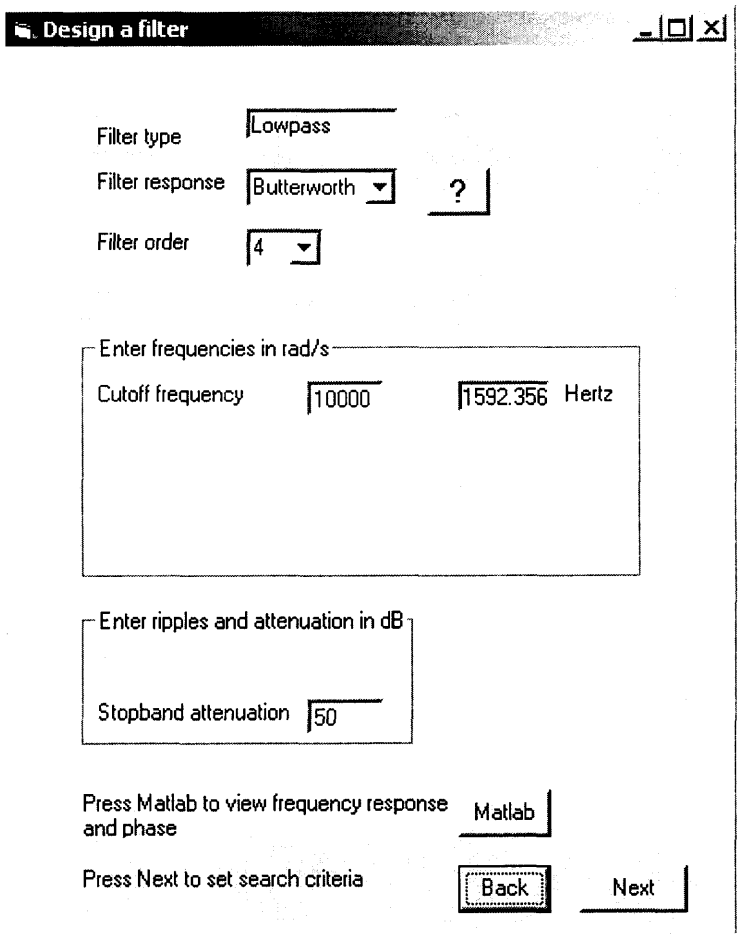


Figure 5.32: *Form for lowpass Butterworth filter of order 4*

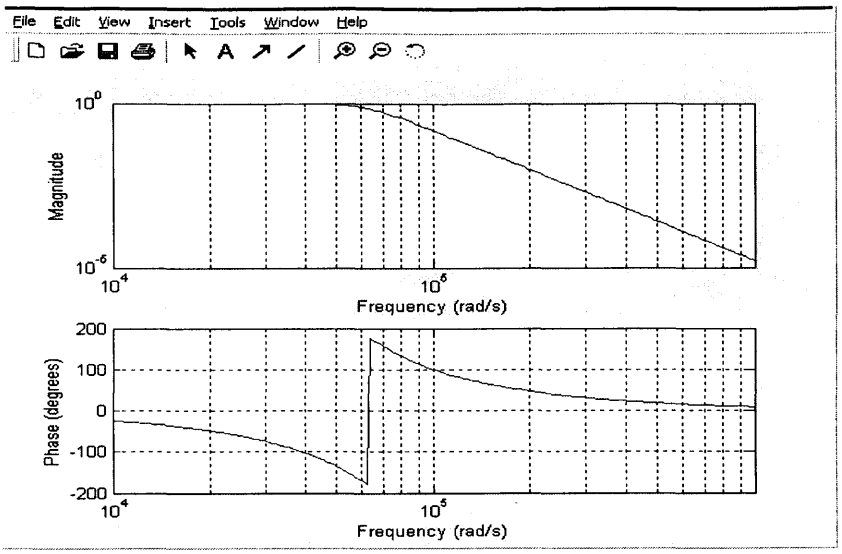


Figure 5.33: *Frequency and phase response using MatLab*

Once the search criteria have been set, suppose the user chose to use the default values, then the result of the search will be to display a list of prospective designs, each with its percentage match as shown in Fig 5.34.

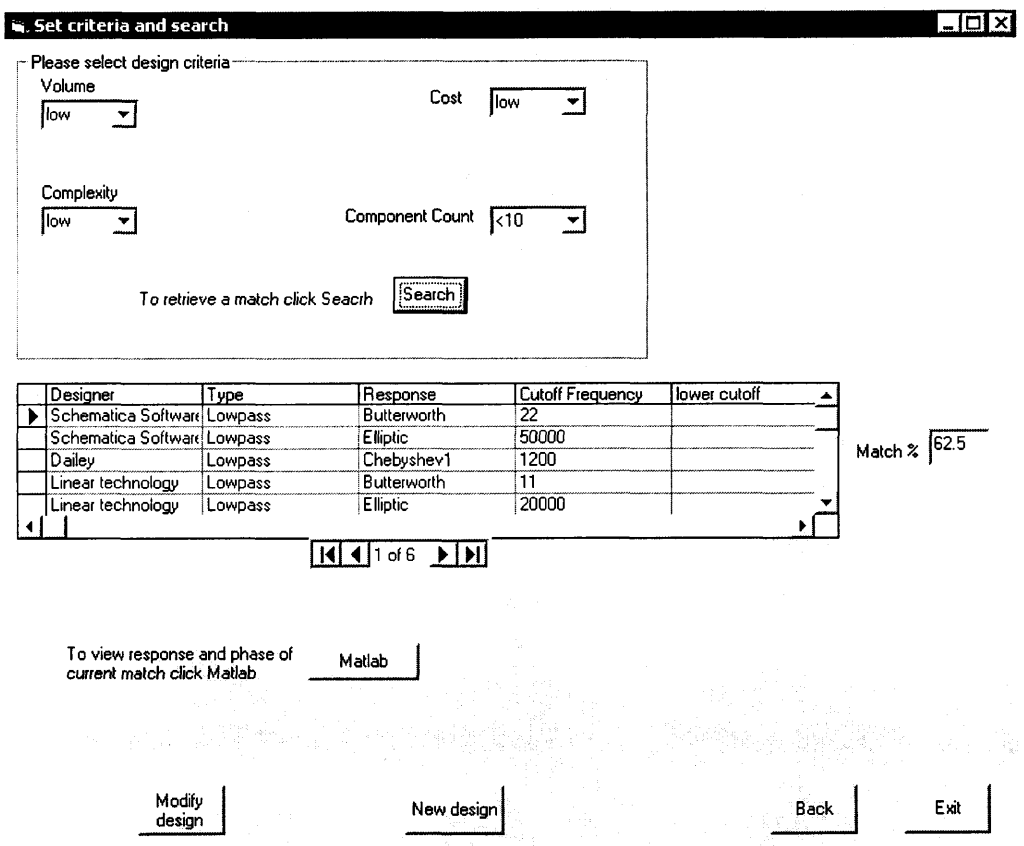


Figure 5.34: *The output search of the design example*

The values for these criteria for the current record are high, low; medium and a component count of 13 respectively. The scores can now be extracted from the tables as:

Volume	Cost	Complexity	Component count
2	10	6	7

The total score is then 2 + 10 + 6 + 7 = 25

The percentage match for this record is then 25/40 ≈62.5 %

At this stage the user can accept, or modify or reject the design. Similarly, if the user decides to use the system by choosing the filter application option the same procedure can be followed.

5.6 Testing

The purpose of testing is to confirming that the system satisfies the stated requirements. As mentioned at the beginning of this Chapter, some early testing was done during the analysis phase described in Chapter 3. This involved the testing of database file to check that each filter design case in the database has completed design fields to ensure that they are consistent and ready to be searched by the inference engine. The other early test was that of checking each of the design cases and making sure that they matched the acquired filter design knowledge.

Once coding has begun, the testing process can proceed in parallel. As each program module is produced, it can be tested individually, then as part of a larger program, and then as part of a larger system. As the implementation of the filter design expert system is based on the design and implementation of a software application, the verification and validation of this expert system is equivalent to the testing stages in building a conventional information system.

Module testing:

In module testing, each module is tested alone in attempt to discover any errors that may exist in the module's code. Referring back to the dialogue diagram of the system (Fig. 5.9), each box can be treated as a module and coded individually. For instance, consider the module '*Select filter type and technology*', as the code for this module is developed it should translate the look and feel of the user interface of this module (see Fig. 5.35) into a functional form in line with the requirements set out in Chapter 4.

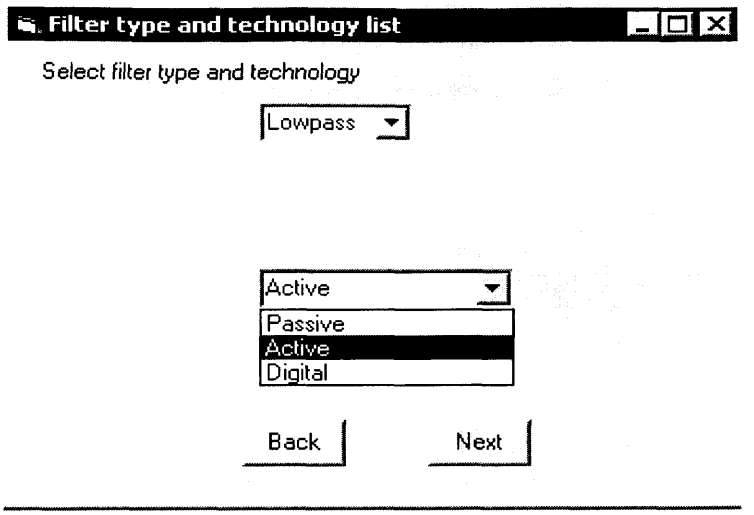


Figure 5.35: *Filter type and technology selection screen*

The code for this form is written in Visual Basic and tested using dummy forms to be accessed via the '*Back*' and '*Next*' commands. The handling of events for each module is shown in Appendix A2 while the Visual Basic code window for the module '*Select filter type and technology*' is shown in Fig. 5.36 as an illustration.

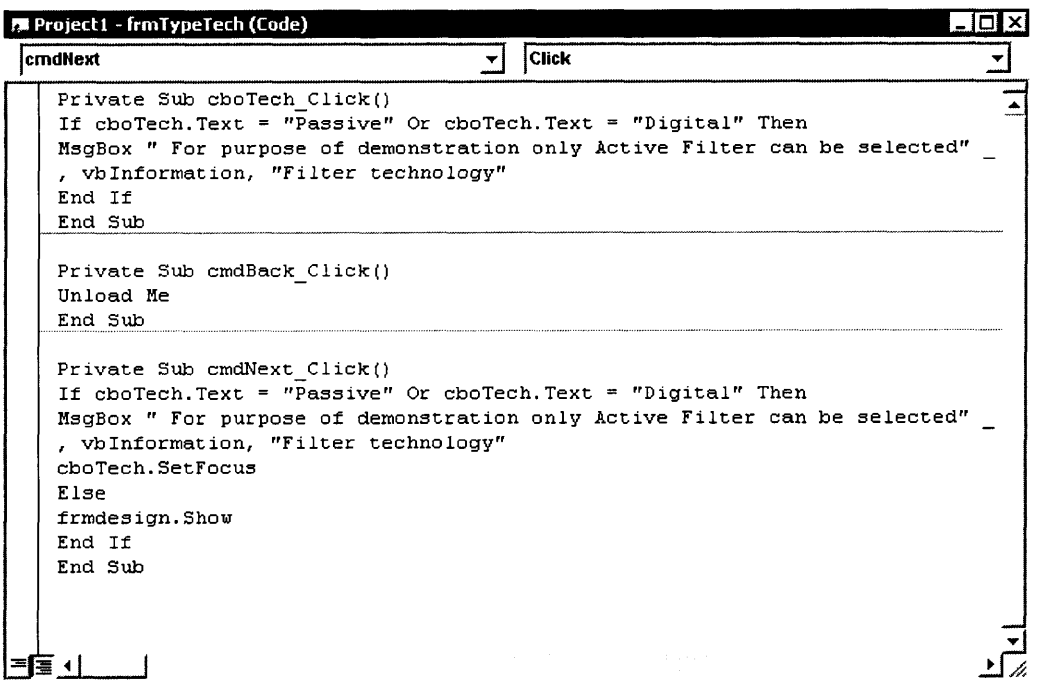


Figure 5.36: *Code window of the* 'Select filter type and technology' *module*

Note that the above was about testing the code and to help in writing correct code. The important concern is to get the code for each module functioned properly before integrating

all the modules into one system. As stated, all the modules discussed in this Chapter have been coded and tested as separate projects. Some of the modules were tested by inputting information from the user and while others were tested when integrated to other modules by passing information between each other.

Integration testing:

Because modules co-exist and work with other modules in programmes and systems, they must be tested in larger groups of related modules in order to complete the overall system test. The process of combining modules and testing them together is called integration testing and is a gradual process. Look again at the dialogue diagram modules. These can be divided into three groups of related modules, the first group representing by the module '*Select filter type and technology*' (Item 2) and its subsequent modules in the hierarchy, Item 2.1, Item 2.1.1 and Item 2.1.1.1. The second group represent the module '*Search by filter application*' (Item 3), where; the third group can be represented by the remaining modules, Item 0 and Item 1 of the dialogue diagram.

Consider the first group involved in integration testing. This consists of four modules on the dialogue diagram that have been coded and tested individually. Start with the top module, '*Select type and technology*' (Item 2) and link it to the next module in the group '*Design entries*'. Theses were tested as a single module and the test involved the functionality of both modules. The other modules '*Searched result*' and '*New design*' were tested with the above modules until they formed the complete group. The group test was based on the design entries and searched the filter design records in the database as well as adding new designs obtained with the aid of external filter design tools.

The next integration test is for the second group represented by the module '*Search by filter application*' (Item 3). As this group contains only one module then the integration test of this group is exactly the same as the test of the module carried out previously.

The last group on the dialogue diagram consists of two modules, the '*Splash screen*' and '*Select the system*'. These modules are seen on launching the system. The test for this group is conducted by checking the first splash screen with information about the system and testing the other module to select filter design from among other options. The next aim is to test the overall system.

System testing:

System testing is a similar process to integration testing but instead of integrating modules into groups for testing, the groups are integrated into systems. The procedure then follows the same incremental process and logic of the integration test. Under both integration and

system testing, not only do individual modules and groups get tested many times, so do the interfaces between modules and groups.

In order to test the system described here, the three groups from the integrating testing have to be linked to conduct a test on the overall system. This starts by launching '*Splash screen*' and '*Select the system*' group. Before adding this group to the other groups, an option form can be added to help user in selecting the appropriate design and search route (see Fig. 5.37). The first option in Fig. 5.37 represents the group of the module '*Search by filter application*' where as the other option represents the group of module '*Select type and technology*' and its subsequent modules.

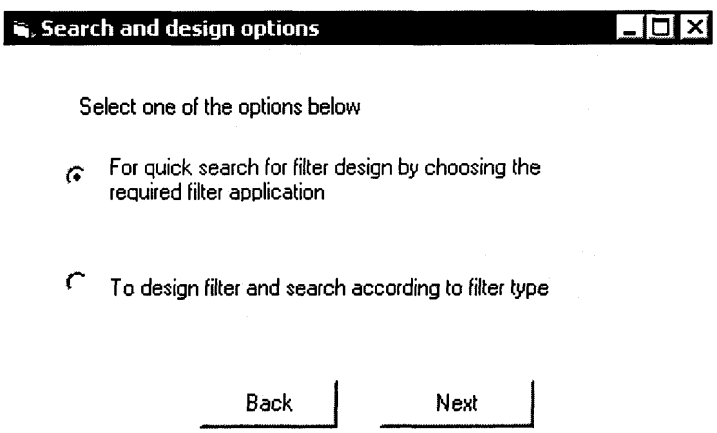


Figure 5.37: *Design options selection screen*

The system test at this stage represents the test of the overall system. The programmer conducts this test in order to verify that the code is correct, the analyst to ensure that the system implementation is what was designed and the user to test the usability of the system.

Usability testing:

The primary goal is to improve the usability of the product being tested. Another is to improve the process by which products are designed and developed. While there can be wide variations in where and how to conduct a usability test, every usability test shares these characteristics[5]:

1. The primary goal is to improve the usability of a product. For each test, it also has more specific goals and concerns articulated when planning the test.
2. The participants represent real users.
3. The participants do real tasks.
4. What participants do and say is observed and recorded.

5. Analyse the data, diagnose the real problems, and recommend changes to fix those problems.

The participants for the usability test will be chosen from disciplines that meet the variety of actual users. Initially three participants were chosen to conduct the test. Each come from a different background of filter design experience and background.

Task Scenarios:

Before starting to conduct the test, it is important to define the task scenarios that tell the participants what is needed to be done during the test. Here are two examples of scenarios:

> Scenario 1:
> *You need to design a 100000 rad/s anti-aliasing filter to be used in an analogue to digital converter circuit.*
>
> Scenario 2:
> *It is required to design a 40000 rad/s, 3^{rd} order, lowpass, Butterworth filter with a stopband attenuation of 45 dB.*

The above scenarios can be handed to the individual participants during the test. The minimum number of participants was to be two; one of them a novice where the other was considered as a filter design expert.

What can be measured?

In usability test, the following can be collected:

Performance measures - Record how much time people take, how many errors they make, how many times they repeat the same error. Most performance measures require careful observation. Examples of performance measures for testing the usability of products include the interface, online help and print documentation.

Subjective measures - These may be either quantitative or qualitative.

Quantitative: Measure something such as accuracy, speed and recall. Accuracy as to how many mistakes user made while using the system. Speed, how long did it take the user to finish the task while recall is about how much does the user remember 15 minutes or 24 hours later.

Qualitative: Use multiple choice and open-ended questions. Multiple choices involve questionnaire while open-ended questions of the form does the user have any suggestions for system improvements? What was the most interesting section in using the system? Why?

Participant 1:

Job: *Research Student*

Nature of work: *Product design*

Filter design experience and background: *Novice*

Computer literacy: *Good*

Participant 1 conducted the test by going through the task scenarios while his moves and reactions where observed when using the system and recorded. During the observation, 100% of attention must be given to the user while noting down everything that can be heard and observed. As a novice, Participant 1 was able to follow the system, though there was an occasional confusion when he moved from one form to another. However, the overall task-based scenarios were achieved with no major difficulties. One of the comments made was to suggest a print command button to print out a report of the design match. Another was to have a summary of the filter specification displayed on the search form. As a novice he found that it is easy and quicker to search for filter design by application rather than using filter specifications or custom design.

Participant 2:

Job: *Network Engineer*

Nature of work: *University network infrastructure*

Filter design experience and background: *Expert*

Computer literacy: *Good*

Participant 2 is graduated with degree in electronic engineering and therefore he considers himself as a filter design expert. When he conducted the test by follow the scenario tasks, the first thing observed was that he was going freely through the different application screens and that it almost seemed as if he had used the system before, most probably because of his filter design knowledge. The first comment about the system he mentioned when asked was " *It is nice layout design for first time users*".

Participant 3:

Job: *Undergraduate student – final year*

Nature of work: *Software engineering*

Filter design experience and background: *Novice*

Computer literacy: *Excellent*

Participant 3 is in 4th year computing and as software engineer is considered as a novice in terms of filter design knowledge. The outcome from using the system was mainly based on the system usability and the interface design criticism. During the test, it was suggested that some command buttons were repositioned on the form. For instance, the command button '*Back*' on the form '*Filter design by application*' to be positioned near to and to the left of the command button '*Exit*'. Also, it was noticed that this form is slightly bigger when compared to the other forms and it was suggested that this should be resized as shown in Fig. 5.38. All these comments were recorded during the test, and immediately after the test was finished the forms were reviewed and changes made accordingly.

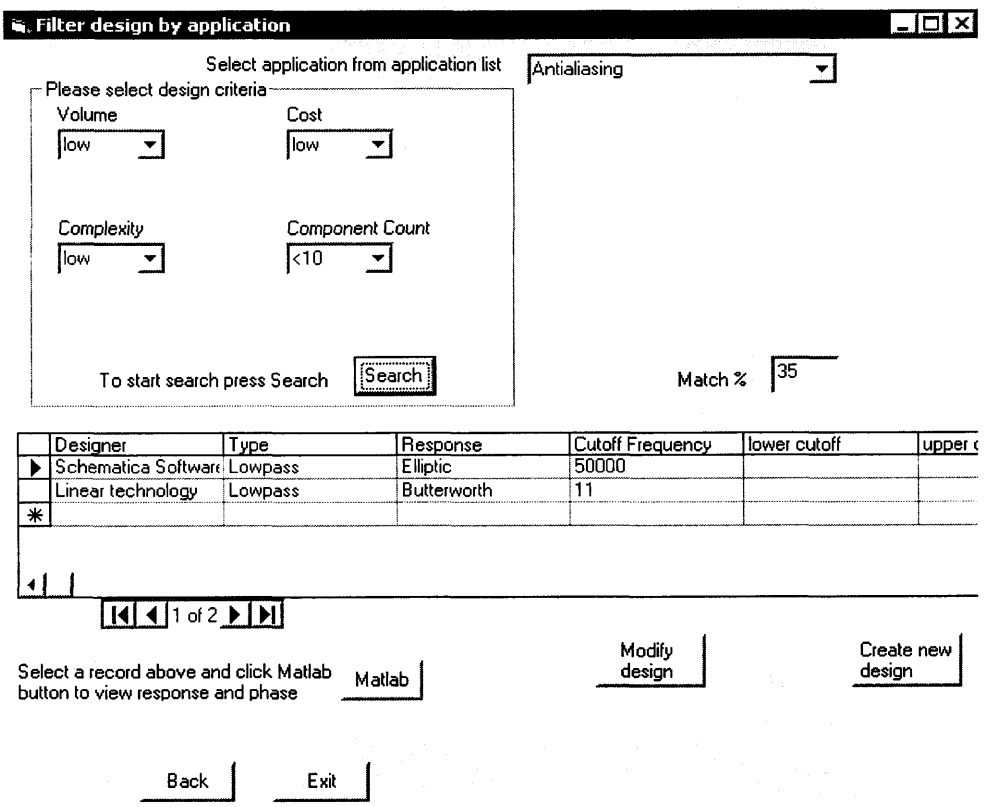


Figure 5.38: *The form after some changes added*

Also during the test, the participants spent a significant amount of time looking for the online help to assist in choosing the filter design parameters. Generally however the participants could follow the system from the perspective of filter design while focusing on the system interface.

Questionnaire:

There are several points during a test when participants are asked questions about a variety of topics; such as their backgrounds, their opinions about a particular task, and their opinions about the overall ease of use of the product. It is necessary to anticipate every question they

might ask and put them into a written form. The main reason for having a written questionnaire is to ensure that every participant is asked the same questions. A sample of questionnaire used is shown in Table 5.6.

The questionnaire in Table 5.6 was given to the participants after they finished the test and they where asked to answer the written questions and add comments. The participants were given time to answer the questionnaire and when these were returned they were reviewed and interpreted along with the data collected from observing the earlier test to identify changes to the system. The comments and the suggestions from participants were fedback into the system and changes were made accordingly.

The usability test is an iterative process, which can take place at early stage when developing the look and feel of the prototype until test the final product. As a result of this simple initial test, most of the changes to the system were in the user interface. For instance, a print command was added, and more online help is needed as at the moment help is for the purpose of demonstration only. In the search form that follows the design form, Participant 1 suggested that before setting the design criteria and starting a search, a summary of the filter specifications should be displayed on the form rather than having to go back to look to the previous form. The comments from the participants about the system context and functional were fairly acceptable in that all were able to follow the system and to perform the tasks that they were asked to do, which was the main objective of the system.

Table 5.6: *Post-Test questionnaire*

Questionnaire

1. How easy or difficult was it to complete the task (circle your answer)

1	2	3	4	5
Very easy	Easy	Neither easy nor difficult	Difficult	Very difficult

2. Do you use the online help to complete the task?

   Yes
   No

3. Using the software was

1	2	3	4	5
Very easy	Easy	Neither easy nor difficult	Difficult	Very difficult

Comment----------

4. Using the software to complete the task was straightforward?

   Yes
   No

5. Was the user interface friendly and easy to follow

1	2	3	4	5
Very easy	Easy	Neither easy nor difficult	Difficult	Very difficult

6. Was it easy or difficult to go and follow the screens?

1	2	3	4	5
Very easy	Easy	Neither easy nor difficult	Difficult	Very difficult

7. Understanding the instruction in the prompts was

1	2	3	4	5
Very easy	Easy	Neither easy nor difficult	Difficult	Very difficult

8. To get what you looking for was it

1	2	3	4	5
Very easy	Easy	Neither Easy nor difficult	Difficult	Very difficult

9. Would you recommend the use of the software to others

   Yes
   No

10. List any other comments you had about the software.----------

5.7 Conclusion

The main conclusions from this Chapter are:

- The implementation of the prototype for the filter expert system designed in Chapter 4 has been achieved. Using an active filter as a prototype, the database file was built containing active filter design cases in a single table comprising records and fields. By coding the user interface designed in Chapter 4, it was linked to the database.
- Visual Basic and SQL are well suited for building the database and the system front end. Visual Basic was used in linking to the database file, which was built using Microsoft Access. SQL commands were used to retrieve data from and update the database using the Case Based Reasoning approach.
- The implemented system is suitable for users who are either novice or expert. As the system is designed to be used for that purpose, users from both levels tested the implemented system. The feedback obtained after the usability test was analysed and the system adapted accordingly. The main responses from the users were that they were able to use the system regardless of their background, they could followed the system and eventually have got what they needed.
- The users can follow the user interface freely for online help and support. At any time the user can get help in a text and graphical form. This could be a general filter design knowledge that might help a novice in understanding filter design, or advanced filter design knowledge that might help the expert.
- The implementation of this expert system prototype is used as a vehicle to implement other filter types such as passive and digital. From Fig. 5.1, a passive filter has no significant difference from the active filter; therefore the implementation would be exactly the same. Whereas the digital filter design as can be seen in Fig. 5.1, has different routes and additional processes, which will add more code to the implementation process. The database design and the coding procedures will be similar to the one used in the prototype.

---

[1] William, A & Taylor, F (1995). *Electronic filter design handbook.* 3rd edition, New York, McGraw-Hill.

[2] Temes & Mitra, (1973). *Modern filter theory and design.* Wiley

[3] French, Michael J., (1990). *Function Costing: A potential Aid to Designer.* Journal of Engineering Design, Vol. 1, No. 1

[4] FilterCAD design tool. Linear Technology Inc.

[5] Dumas & Redish. (1999). *A practical Guide to Usability Testing*. Intellect Ltd.

Chapter 6

Linking to Other Electronic Domains and the Overall Mechatronics System

6.1 Introduction

Having developed the filter design expert system, the same techniques can be used to develop similar systems for other electronic sub-systems such as amplifiers, microprocessors, DSPs and microcontrollers.

In the case of amplifiers, which, unlike filters, exist only for analogue signals, the hierarchy tree would have the form of Fig. 6.1. The lowest level of the hierarchy, though not shown in the diagram, will be concerned with the details of the amplifier specification such as gain, input resistance, output resistance, bandwidth and so forth. Once the hierarchy is established, knowledge can be acquired using human experts, texts and other sources to create the database of designs.

As with filters, the amplifier design system would have links to appropriate design and simulation tools to aid users in modifying or developing a design.

Other electronic components such as DSPs, microcontrollers and microprocessors, contain both software and hardware elements and are used as implementation technologies for sub-systems such as digital filters. As these devices are programmable, the emphasis here will be on the choice of appropriate hardware followed by the development and implementation of the required software. In this case, a user would expect to be able to get help and advice when implementing a design using a DSP, microcontroller or microprocessor both on the choice of the appropriate hardware and on the software implications. This is likely to involve a different strategy from that for filters and amplifiers, which would need to take account of other related constraints on the implementation process. For instance, if a microprocessor or microcontroller based solution was to be developed, then it would be necessary to utilise facilities already available to the whole system under design and not have to purchase them specially. Software implications might be the choice of programming language and the compilers used in writing the code such as 'C' and assembly language for the chosen hardware implementation. Therefore the hardware and the software choices are constrained on each other.

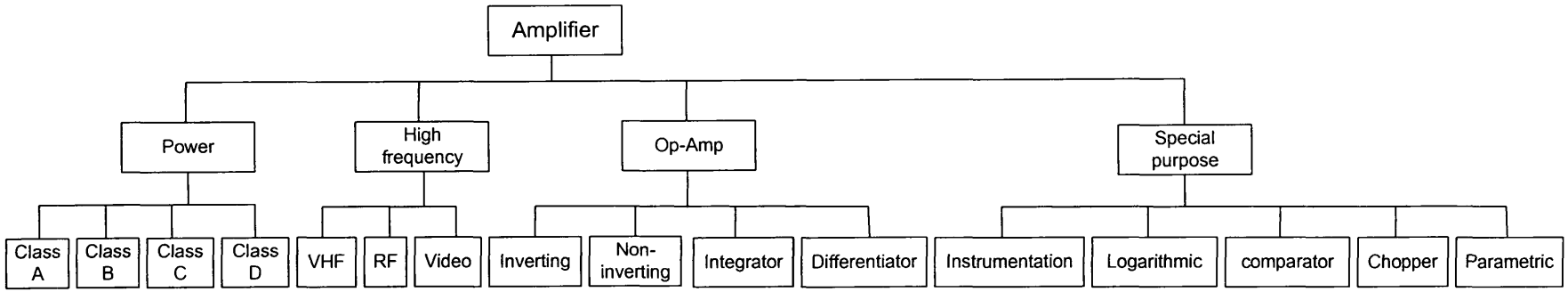


Figure 6.1: *A hierarchy tree for an amplifier*

The aim of this Chapter is to consider and define ways of linking together the individual modules and sub-modules resulting from the decomposition of the mechatronics system as presented in Chapter 1, into its different technologies. With the aid of appropriate expert system strategies, these links can be achieved by setting constraints on each sub-system, which in turn will be related to the other sub-systems when undertaking a design.

In order to proceed it is important to discuss how the elements of a mechatronics system are related. The first part of the Chapter is introductory and defines the basis on which the different mechatronics components are interfaced. The second part of the Chapter examines the use of the Data Dictionary as the means of defining the interconnections between system elements and as a means of constraint transfer between processes. Also, some of the introductory materials and figures in this Chapter were mentioned in Chapter 1 and used again here for the purpose of clarification.

Finally, there will be a specific design example identifying the links between the electronic sub-modules as a means of illustrating the process, which will then be extended to encompass the links to other system components at all levels required by a mechatronics system.

6.2 Mechatronic Component Interfacing[1]

In mechatronic systems, a computer is interfaced to signal and power transmission components. Signal transmission components may have analogue outputs as in Fig. 6.2 or digital outputs as in Fig. 6.3. Normally, signal transmission components interfaced to the processor are transducers forming part of a data acquisition system, while power transmission components interfaced to a PC are actuators receiving command signals from the processor.

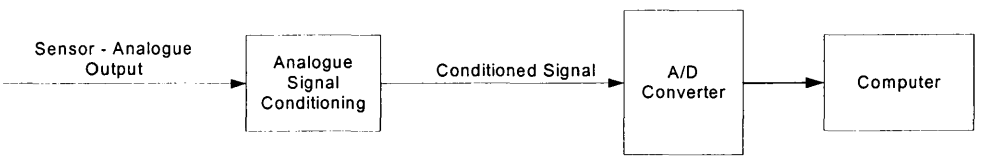


Figure 6.2: *Signal conditioning of an analogue sensor*

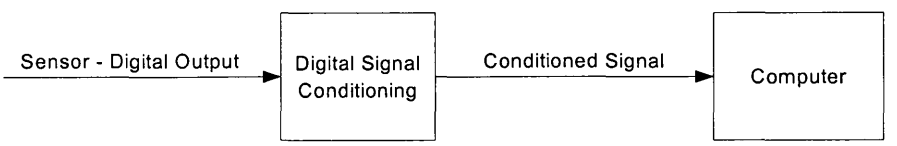


Figure 6.3: *Signal conditioning of a digital sensor*

Power transmission components can receive analogue commands as in Fig. 6.4 or digital commands as in Fig. 6.5. Output signals from the processor have to be compatible with the actuator input requirements. This usually implies a signal conversion to the voltage and current demanded by the actuator.

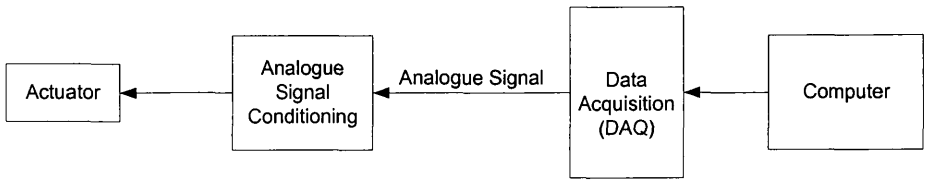


Figure 6.4: *Actuator interfaced with a PC*

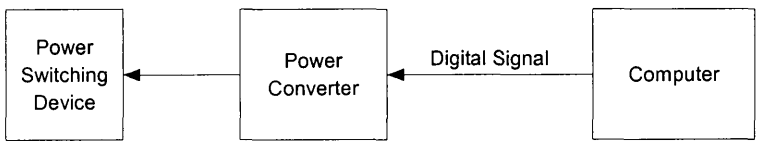


Figure 6.5: *Digital command of a power switching device*

The above diagrams can be combined to generate the schematic diagram of Fig 6.6 for the computer-based control of a process, or of the entire mechatronic system. The thin arrows and lines in Fig. 6.6 correspond to signal flow while the broad arrows correspond to power flow.

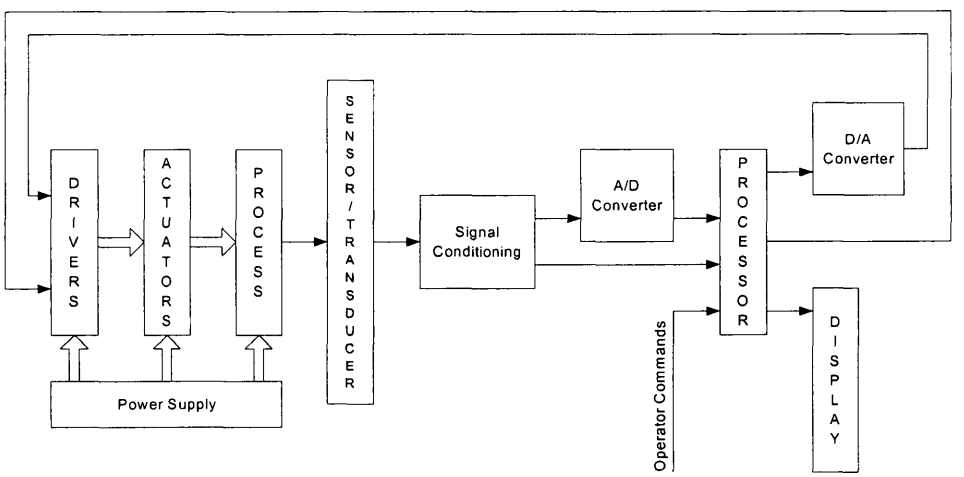


Figure 6.6: *Schematic diagram of computer based monitoring and control of a process*

Referring to Fig. 6.6, the process variables, as measured by the sensors/transducers, are signal conditioned and converted if necessary from analogue to digital form and transmitted to the processor. The processor performs real-time monitoring and control as well as signal analysis and provides the actuator commands and the necessary system monitoring and display.

The successful design of such a system requires the combination of the designs for the individual system components and sub-assemblies of Fig. 6.6 in order to form the overall system. The designer must carefully consider the compatibility between the individual components when proceeding with the design. For example, in Fig 6.6, it may be required to interface the analogue output from a transducer, for instance in the range 0 to 100 mV, to the analogue to digital converter that may have an input voltage in the range 0 to 5 volts. The designer should then realise that the signal conditioning must be in the form of an amplifier with a gain of 50 to position the signal in the required range.

It is therefore clear that the design of any element of the mechatronic system of Fig 6.6 is constrained by the inputs to and the outputs from it to and from other system components. In order to understand this, the diagram of 6.6 can be represented by a fully decomposed Data Flow Diagram (DFD) as a generic mechatronics system formed from implementable function components as shown in Fig. 6.7. The Data Dictionary and the process specification are used when implementing any of the processes. For instance, the designer is required to define each flow specified in the data flow diagram in a Data Dictionary. When the design is subsequently implemented, these flows represent the interface between the different system elements.

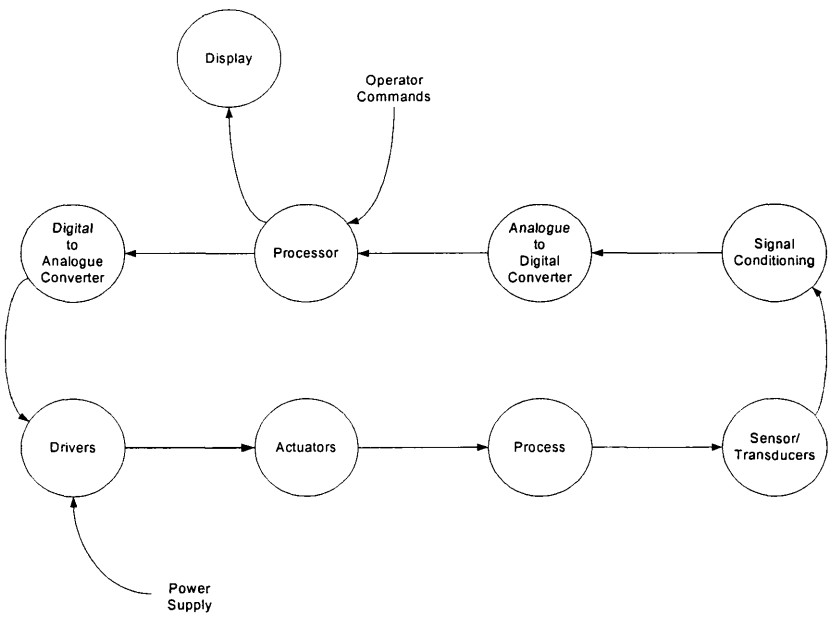


Figure 6.7: *Data Flow Diagram of computer based monitoring and control of a process*

6.3 Data Dictionary in Design

The Data Dictionary can be used not only to indicate the interconnection between processes, but also as a means of constraint transfer between the processes[2]. As the system is functionally decomposed, the design requirements are used to define each of the flows specified in the Data Flow Diagram (DFD) in the Data Dictionary. These flows represent the links and the interfaces between the different system process functions. It is therefore required that the constraints imposed at either end of the flow should be consistent. In this way the flows can be used to propagate constraints as well as information round the system.

The contents of the Data Dictionary, as discussed in Chapter 4, are made up of definitions such as, '*name of flow*', '*type of flow*', '*source and finish nodes*', '*type of signal*', '*constraint*' and so forth. As the interest here is on processes interfacing, then the emphasis will be on '*signal type*' and '*constraint*'.

Signal type

This field is used to define to the system the form of the signal by which the contents of the flow are to be represented. It is not proposed that the designer should necessarily define the type of signal to be used for flows in general, except where there is a requirement to have a particular form of representation governed by external constraints[3].

The three common signal types are analogue, digital and power. For instance, the active filter design used as a prototype in Chapter 5, was of the analogue signal type. A digital signal is used in the output of a digital sensor or at the input or output of a digital controller, whereas the power signal type is used where an actuator is concerned.

Constraints

This field contains a list of bounding values placed on the flow. The signal type field will cause a list of relevant criteria to be displayed, from which the appropriate constraints may be selected. For example, if the signal type is defined as analogue voltage, the constraints could be:

- Signal voltage
- Offset voltage
- Source/ load/ input or output resistance
- Frequency range

Once the constraints are defined, they can be assigned values, which may then be defined as the minimum, maximum or nominal values of that constraint. A minimum would allow for

anything greater to be accepted, and maximum would allow for anything less to be accepted, and a nominal value would be that which is used in calculations, but from which variations may occur[4].

An example of using the data dictionary flows as a means of constraints transfer between processes is seen in the following simple example[5]:

> *If the output of process A in Fig. 6.8, has an amplitude of 1 volt, a bandwidth of 20KHz and an output resistance of 1 KΩ then these constraints could be transferred along the flow to the destination process B, constraining that to have a bandwidth of 20 kHz; be able to accept an input of greater than 1 volt; and in order to buffer the output of the previous stage, a source resistance of greater than 10 KΩ.*

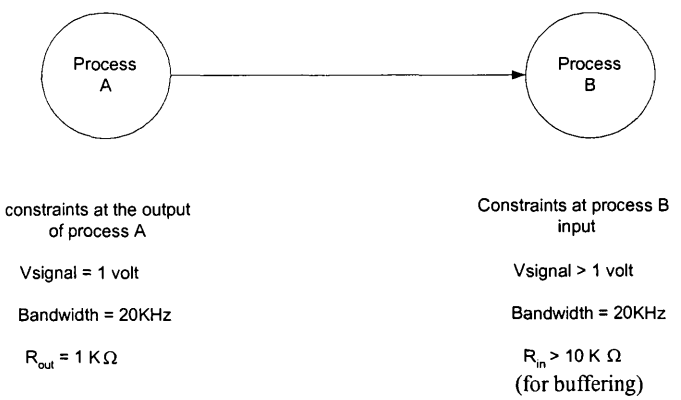


Figure 6.8: *Constraints in interfacing process A and process B*

6.4 Design Example[6]

Although this design example is simple in itself, the approach adopted will form the basis of a methodology, which will support the integration of such simple sub-systems to form a large complex system, and therefore it will be discussed in some detail in order to provide the necessary context.

The concept of closed-loop control is widely applied in industry, particularly in the process sector. One such application is the control of the water level in a tank, where the requirement is that the inflow into the tank must be automatically adjusted in line with the outflow. The basic hardware needed for the control of the system is:

1. Sensing system
2. Controller
3. Actuator
4. Tank
5. User input

The sub-system diagram is shown in Fig 6.9, illustrating the interrelationship between the various hardware elements from which it is seen that the overall system is formed as a combination of a number of sub-systems.

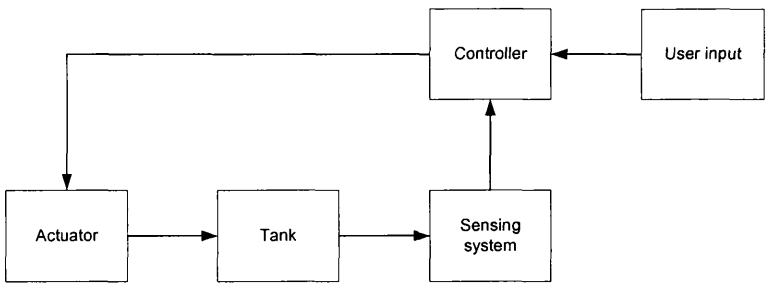


Figure 6.9: *The block diagram for level control loop*

The designer at this stage can proceed with the design with the aid of the design expert system. Consider that any action taken by the designer is treated as a manual option, whereas any automatic option is one to be performed by the design expert system (or *DES*). The aim of this is to aid the designer in developing the overall design and which draws upon individual system modules such as that for filter design.

Suppose that the context diagram of the water level control system along with its process specification and Data Dictionary are completed with the aid of the *DES*. It is now possible to commence the decomposition of the design using data flow diagrams. The first level of the decomposition is completed by considering the main component parts required by the system, as represented in the block diagram and redraw as shown in Fig. 6.10. The process specification for each process and the Data Dictionary of the flows completed by the designer with the aid of the *DES*.

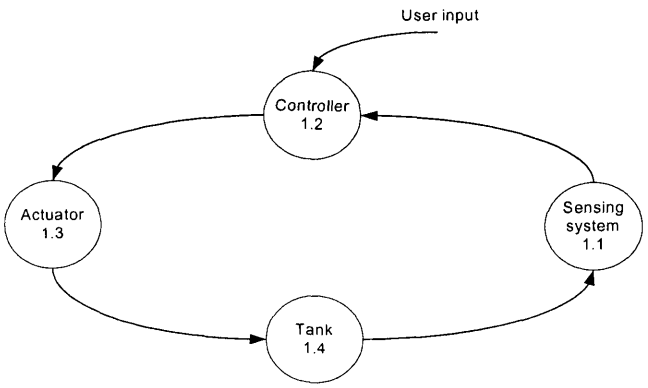


Figure 6.10: *The first level decomposition for the water level control system*

Assuming that the controller is to be implemented digitally; this means that the input to the sensing system is an analogue quantity (level) and the output to the controller is a digital signal.

Consider the sensing system process in more detail. As shown in Fig. 6.11 this has input and an output flows defined in the Data Dictionary. The source to the sensing system process is the tank process and the destination is the controller process.

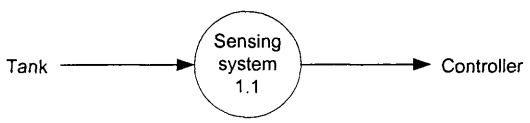


Figure 6.11: *The sensing system process*

As the input to the sensing system process 1.1 is an analogue signal represented by the water level in the tank, then the designer recognised the need for an analogue sensor. Once the sensor system process is selected, a process specification window would appear automatically to ask the designer to enter the process name and to choose the implementation function for the sensor process from a list of functions. In this case the designer chooses '*type*' level sensor but does not at this stage opt to select a specific sensor from those available.

The designer can now review the output flows, the Data Dictionary for '*type*' level sensor and for any other processes linked to sensor process. The input and the output flows are then

as represented by Table 7.1. A question mark is used in the table to indicate that a value is not as yet known, while bold and italicised text is used to indicate new entries[†].

Table 6.1: *The input and output flows of the sensor and controller processes*

	Sensor		Controller
	Input flow	**Output flow**	**Input flow**
Source	***Tank***	***Sensor***	***Sensor***
Name	***Sensor input***	***Sensor output***	***Sensor output***
Type of flow	***Information***	***Information***	***Information***
Destination	***Sensor***	?	***Controller***
Type of signal	***Analogue***	***Analogue***	***Digital***
Signal voltage	?	?	?
Frequency range	?	?	?

Once a sensor is chosen, then the signal type in the input and the output flows of the sensor process will be added automatically by the *DES*. In the case of the level sensor these will be of type analogue. The *DES* will then realise from checking the type of signal in the Data Dictionary that there is a conflict and may then come up with a warning message of the form of Fig. 6.12.

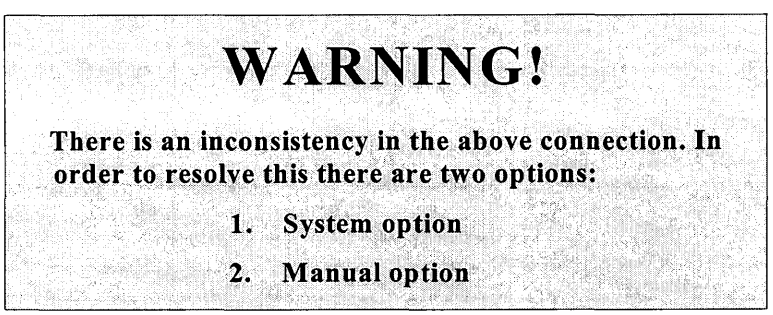


Figure 6.12: *DES's warning message to the designer*

If the designer chooses the system option, this means that the *DES* will resolve the problem by adding an analogue to digital converter (A/D) between the sensor and the controller. If the manual option is selected, the *DES* will suggest to the designer alternatives such as use an analogue controller or a digital sensor.

Suppose that the designer chooses the system option, then the system will automatically insert an A/D converter into the process as in Fig. 6.13 and automatically open the process specification window where the designer will name the process and choose the appropriate implementation function, in this case an A/D converter.

[†] This convention is used throughout the Chapter.

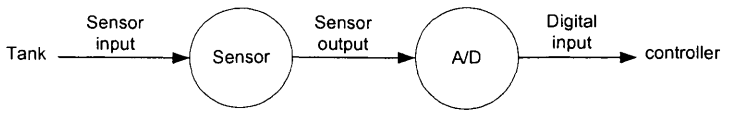


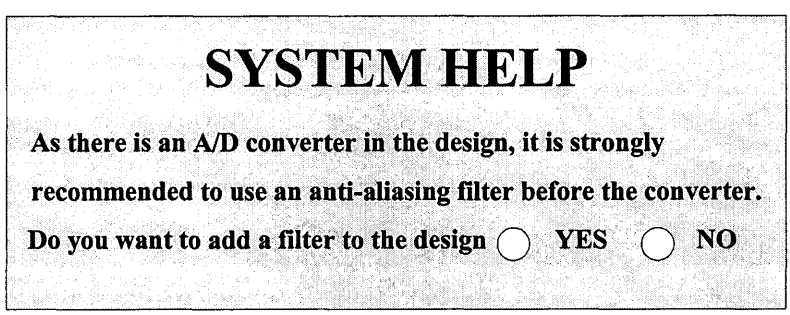
Figure 6.13: *The input and the output to the processes 'Sensor and A/D'*

The Data Dictionary entries of the A/D process input and output flow will be recorded automatically by the *DES* and some of the empty entries in other process flows will be modified accordingly. The Data Dictionary table will now be as shown in Table 6.2.

Table 6.2: *The updated input and output flows of the sensor and A/D processes*

	Sensor		**A/D**	
	Input flow	**Output flow**	**Input flow**	**Output flow**
Source	Tank	Sensor	***Sensor***	***A/D***
Name	Sensor input	Sensor output	***Sensor output***	***Digital input***
Type of flow	Information	Information	***Information***	***Information***
Destination	Sensor	***A/D***	***A/D***	***Controller***
Type of signal	Analogue	Analogue	***Analogue***	***Digital***
Signal voltage	?	?	?	?
Frequency range	?	?	?	?

For the above arrangement the *DES* will intelligently realise that there is a need for an anti-aliasing filter and will generate the following prompt.



If the designer selects YES, the system will automatically add a filter process between the sensor and A/D as in Fig. 6.14.

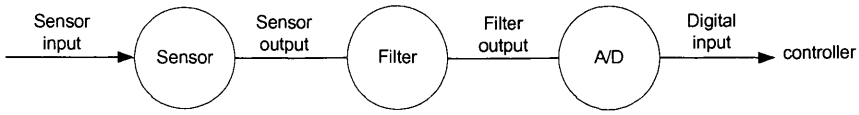


Figure 6.14: *The input and the output to the processes 'Sensor, filter and A/D'*

As before, once the filter process is inserted, the Process specification window will appear automatically and other process specifications will be updated by the *DES* to produce Table 6.3.

Table 6.3: *The input and output flows of the sensor, filter and A/D processes*

	Sensor		**Filter**		**A/D**	
	Input flow	**Output flow**	**Input flow**	**Output flow**	**Input flow**	**Output flow**
Source	Tank	Sensor	***Sensor***	***Filter***	***Filter***	A/D
Name	Sensor input	Sensor output	***Sensor output***	***Filter output***	***Filter output***	Digital input
Type of flow	Information	Information	***Information***	***Information***	Information	Information
Destination	Sensor	***Filter***	***Filter***	***A/D***	A/D	Controller
Type of signal	Analogue	Analogue	***Analogue***	***Analogue***	Analogue	Digital
Signal voltage	?	?	?	?	?	?
Frequency range	?	?	?	?	?	?

The designer can now use the filter design expert system to design the requisite filter with the constraints being transferred by the *DES* to identify that the filter required is an anti-aliasing filter and must therefore be of *type* analogue.

At this stage the filter design expert system will come up with a list of an anti-aliasing filter designs, each with different specification also the designer can go back to the *DES* to provide more information about the required filter design such as the cut-off frequency.

SYSTEM HELP

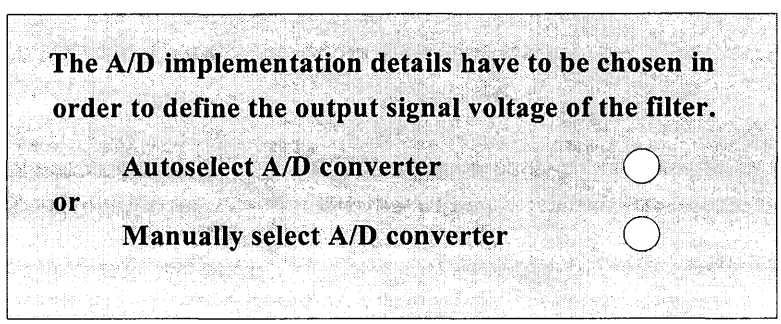
In order for the designer to choose the appropriate filter then there is a need to define the filter specifications from the *DES* Data Dictionary.

At the Data Dictionary, the cut-off frequency of the filter can be determined from knowledge of the frequency range of output signal voltage of the sensor. This requires the designer to return to the implementation database to find an appropriate implementation. Suppose the choice is for a sensor with a signal voltage of 10mV and frequency range of 25 to 100Hz. These specifications are fed into the Data Dictionary where the related entries will be updated as shown in Table 6.4.

Table 6.4: *The Updated input and output flows of the sensor, filter and A/D processes*

	Sensor		Filter		A/D	
	Input flow	**Output flow**	**Input flow**	**Output flow**	**Input flow**	**Output flow**
Source	Tank	Sensor	Sensor	Filter	Filter	A/D
Name	Sensor input	Sensor output	Sensor output	Filter output	Filter output	Digital input
Type of flow	Information	Information	Information	Information	Information	Information
Destination	Sensor	Filter	Filter	A/D	A/D	Controller
Type of signal	Analogue	Analogue	Analogue	Analogue	Analogue	Digital
Signal voltage	?	***10mV***	***10mV***	?	?	?
Frequency range	?	***25-100Hz***	***25-100Hz***	?	?	?

Suppose there is now a match in the form of a passive anti-aliasing filter with a 100Hz cut-off frequency and an attenuation of 5. When these specifications are fed-back the *DES* will come up with the following message:



Suppose the choice is of an A/D converter with an input voltage of 0 - 2.5V and a sampling frequency range of up to 1,000Hz and 8-bit resolution. The Data Dictionary is then as shown in Table 6.5.

Table 6.5: *Input and output flows of the sensor, filter and A/D processes*

	Sensor		Filter		A/D	
	Input flow	**Output flow**	**Input flow**	**Output flow**	**Input flow**	**Output flow**
Source	Tank	Sensor	Sensor	Filter	Filter	A/D
Name	Sensor input	Sensor output	Sensor output	Filter output	Filter output	Digital input
Type of flow	Information	Information	Information	Information	Information	Information
Destination	Sensor	Filter	Filter	A/D	A/D	Controller
Type of signal	Analogue	Analogue	Analogue	Analogue	Analogue	Digital
Signal voltage	?	10mV	10mV	***2 mV***	***2.5V***	***2.5V***
Frequency range	?	25 -100Hz	25 -100Hz	100Hz	?	?

The following message now results. The amplifier and the other processes are shown in Fig. 6.15 and the updated Data Dictionary entries in Table 6.6.

To implement the system using the designed passive filter there is need to amplify the input signal to reach the A/D process input of 2.5V. At the moment it reaches that point with a level of 2mV. The system suggests inserting an amplifier to amplify the signal voltage to the required level

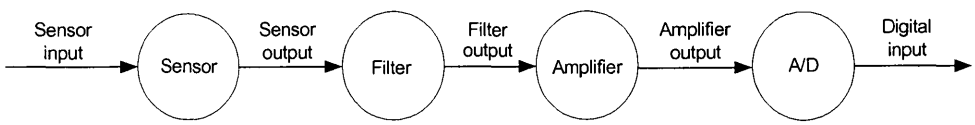


Figure 6.15: *The input and the output to the processes 'Sensor, filter, amplifier and A/D'*

Table 6.6: *The input and output flows of the sensor, filter, amplifier and A/D processes*

	Sensor		Filter		Amplifier		A/D	
	Input flow	**Output flow**	**Input flow**	**Output flow**	**Input flow**	**Output flow**	**Input flow**	**Output flow**
Source	Tank	Sensor	Sensor	Filter	***Filter***	***Amplifier***	***Amplifier***	A/D
Name	Sensor input	Sensor output	Sensor output	Filter output	***Filter output***	***Amplifier output***	***Amplifier output***	Digital input
Type of flow	Info.	Info.	Info.	Info.	***Info.***	***Info.***	Info.	Info.
Destination	Sensor	Filter	Filter	***Amplifier***	***Amplifier***	***A/D***	A/D	Cont-roller
Type of signal	Ana.	Ana.	Ana.	Ana.	***Ana.***	***Ana.***	Ana.	Digital
Signal voltage	?	10mV	10mV	2mV	***2mV***	***2.5V***	2.5V	2.5V
Freq. range	?	25 -100 Hz	25 -100 Hz	0 -100Hz	***0 -100 Hz***	***0 -100Hz***	?	?

The gain of the amplifier will be calculated by the *DES* as 1250. This gain is quite high and in order to implement it the *DES* may suggest that a two stage-amplifier with gains of (say) 100 and 12.5 is used or that an active filter is substituted for the passive filter.

6.4.1 Design Impact on All Sub-Systems Levels

Having completed the design of the '*Sensing system*' element, it can be considered now as a part of a complex mechatronics system along with other sub-systems such as '*Controller*' as shown in the hierarchy of Fig. 6.16. For the purpose of clarification, and instead of representing a large complex system using many DFDs, one hierarchical top-down diagram can be used which shows all different layers of the level 1 sub-system '*Water Level Controller*'. Though the level 1 sub-system 1.2 and 1.3 in the diagram are not identified here, they are included to demonstrate how the complex system is formed. Similarly, the level 3 sub-systems associated with the level 2 sub-systems '*Controller*', '*Actuator*' and '*Tank*' are excluded from the diagram.

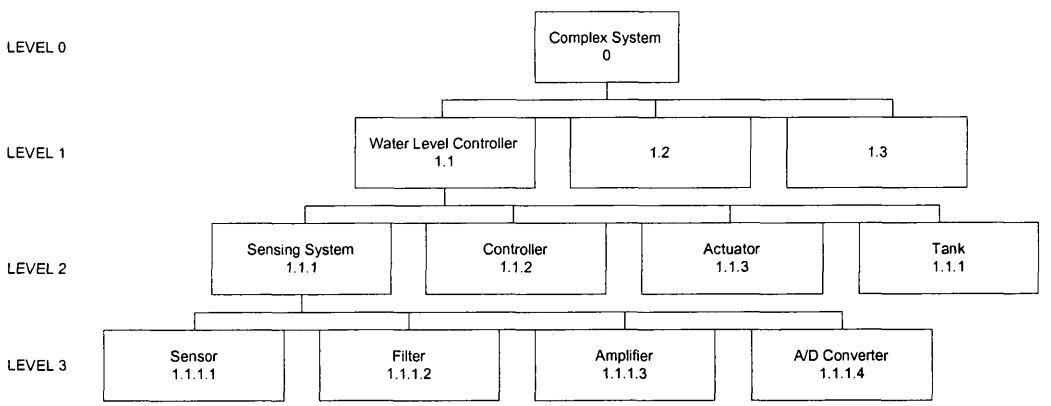


Fig. 6.16: *A water level controller part of a complex system hierarchy*

As design is an iterative process, the other sub-systems in level 2, namely '*Controller*', '*Actuator*' and '*Tank*' could be designed and implemented in exactly the same way as the '*Sensing system*'. Though the sub-system '*Tank*' might be different as this is non electronic and not a mechanical assembly and therefore its design might be based on other factors such as the components attached to it like valves and water pumps along with its physical size, input and output flows and so forth. By considering the design and implementation of all the sub-systems in level 2 and their associated sub-systems in level 3, as these are completed then the next step is to move up to level 1 where sub-systems 1.1, 1.2 and 1.3 can be designed and implemented in a similar manner using the Data Dictionary to manage constraint transfer until the overall complex system at level 0 is completed.

Having developed the design of the overall system, the question now is what would happened if the designer inserts different design parameters for any of the sub-systems at any level, and what is the impact of this on other sub-systems and levels.

This question can be answered by considering that the designer did make a change to the design of one of the sub-systems in level 3, for instance, the '*Amplifier*'. The *DES* will automatically detect the change and update the Data Dictionary. As the sub-system '*Amplifier*' is connected directly to sub-systems '*Filter*' and '*A/D Converter*', their Data Dictionary entries will also be affected and updated. As the '*Filter*' sub-system is attached to the sub-system '*Sensor*', then its Data Dictionary will also be updated. Thus, the change that occurred to the Data Dictionary flows for '*Amplifier*' is propagated to other sub-systems at the same level and elsewhere.

Moving to the next upper level, level 2. The sub-system '*Sensing system*' is the parent for level 3 sub-systems; therefore its Data Dictionary is updated by the *DES*. This sub-system is connected to the next sub-system '*Controller*' through the sub-system '*A/D Converter*' therefore its Data Dictionary entries will be flagged and updated, and similarly with the sub-systems '*Actuator*' and '*Tank.*' All these changes can then be propagated up to the parent sub-system '*Water Level controller*' at level 1, where its Data Dictionary flows will be updated by the *DES* in a similar way.

From the example above it is concluded that the change in any of the sub-systems is propagated to other sub-systems in either direction and to their parents in the upper levels.

The same approach can be applied to any sub-system at any level for instance; a change to a sub-system at a higher level changes the Data Dictionary, which will then propagate down to the lower levels.

6.5 Summary

The following results have been established in this Chapter:

- The developed filter design expert system is a vehicle to develop other design expert systems.
- The impact resulting from linking the design of electronic sub-systems with other electronic domains has been defined. As was seen in the water tank example, the input and output flows in the Data Dictionary of processes under design are filled automatically by the system and updated at each time new design information appears. Each of theses processes may be supported by an associated expert system. Considering constraints on and from each process, it has been found that if the processes are interfaced appropriately then the system will proceed through the design automatically and intelligently. If there is an inconsistency between processes, the system it will prompt the designer through a message identifying the conflict and suggesting remedy. When needed, the designer will have access to any of the expert systems and can feed information from the system into this and back to the higher level system *DES*, which will be updated accordingly.
- The impact of linking the design of any sub-system to all other sub-systems at all levels has also been identified. Any change in any sub-system design will flag changes in other sub-systems, either connected directly or as part of a network.
- The example of water tank control was a simple system but it constitutes an overall complex system. Therefore, the design of a complex system is an integrating and iterative process based on the design of the simple sub-systems.
- The designer may choose to use the support provided by the system by letting the system do as much as it can with minimum intervention. The designer only intervenes when asked by the system to choose an option or take a decision in relation to the design. The system is itself designed to intelligently interact with the designer. The role such a computer system is to assist the user and to complement the strengths and skills of the human designers.

---

[1] Dan Necsulescu, (2002). *Mechatronics*, Prentice hall, USA

[2] R.M. Walters, (1996). *The high-level design of electronic systems*. PhD thesis, Lancaster University.

[3] ibid., P207.

[4] ibid., P209

[5] ibid., P161

[6] Fraser C, Milne J.(1994). *Integrated electrical and electronic engineering for mechanical engineers*. McGraw-Hill

[7] ibid., P501

Chapter 7

Conclusions

The overall aim of the thesis as presented in Chapter 1 was to research and define means by which design tools and methods can be developed to assist in the design of a mechatronics system. This is based on the functional decomposition of a mechatronic system at top the level and the development of an expert system as a support tool, initially for the electronic domain. It was also intended to define the ways in which the system can manage the transfer of information, including constraints and conflicts, between levels and domains within the mechatronic design process.

Because of the diversity of electronic systems and their complexity, one was chosen as an exemplar, namely filters. The analysis, design, implementation and testing of a filter expert system was completed in the thesis, and it is argued that the same techniques can be applied to other electronic systems such as amplifiers. The linking of the electronic sub-system to other sub-system designs using the Data Dictionary for constraint management and transfer was identified in Chapter 6.

This Chapter concludes the work by reviewing the statement of the problem as set out in Chapter 1 and summarises the findings of this research. The Chapter also summarises the thesis contributions and finally, uses the experience and understanding gained to suggest a programme for future research.

7.1 Functional Design of Mechatronic System

Functional decomposition based on Data Flow Diagrams is used successfully in the design of mechatronic system and its components. This decomposition provides the designer with the ability to breakdown the mechatronic system specification into its functional elements. The designer is then able to proceed through successive levels of decomposition until the system cannot be further decomposed, indicating a possible implementation for a defined sub-system function.

The external communication between these functions is defined in the Data Dictionary and the description of the specified process. The Data Dictionary plays an important role not only in the interconnection between processes but as a means of management the transfer and propagation of constraints.

Also, as shown in Chapter 6, functional decomposition can be applied not only for small systems but also for complex ones. This was shown in the design example in which a complex system is broken down into its component sub-systems in a hierarchical manner. As design is an iterative process, the design of a complex system can be treated as the integration of the individual sub-systems in a top-down, bottom-up approach.

7.2 The Expert System for Electronic Design is Viable

The required expert system for electronic design within the context of mechatronic as set in Chapter 1 has been successfully implemented. The system has a user interface, which has been designed to suit users who are either novices or experienced in electronic design. The system after being implemented has been tested by a few users with various levels of design experience to improve system usability.

The filter design exemplar was an appropriate choice for developing the electronic design expert system methodology. As prototyping is an effective paradigm for expert system development, and because of the similarities in the basic design principals across filter types, it was decided to first build a prototype of the active filter design expert system. Once a robust system has been developed for active filters the prototype can then be expanded to handle all filter types.

Any user from the mechatronics design team would be able to freely use this expert system to guide them through the design process and search the database for a design fit. The system was designed in an interactive way such that the user can move forward and backward between system windows, making the application usable. Users are thus able to follow the filter design steps in an interactive way. The user can also get any help needed in relation to filter design, this help can be accessed at any time that the user might required. The user can also review at any time the frequency and phase response of the current design.

The other main purpose of the system is to identify a design fit in the database. Such a database has been successfully built in the course of this project. To search the database there is a need for the user to set the criteria that the search will be based on. The design criteria fields in the filter design database table such as '*Cost*', '*Component count*', '*Volume*' and '*Complexity*' were used to set design preferences which can then be used when searching for a design match. Although it was decided to use crisp sets in the thesis for the reasons discussed in Section 3.5.1, the use of fuzzy sets has been evaluated and the importance and possible impact of the technique on any future version of the system identified. It is therefore a high priority element of any future implementation and one,

which might be considered a project in its own right to design and implement a fuzzy expert system.

The system database is an important of the case based reasoning and the search process to establish a design match or fit. Should the user not find any suitable design in the database, then with the aid of the system, and specifically the filter design tools linked to the application, the user can develop the required design. The user might find that the best fit is close to the required filter design and accept it as it is or, it not be fully satisfied, can adapt or modify the design. The user may also decide to take the design to a human expert if available and ask for help in adapting the design. Any modified design will be stored in the database as a new design, and of course the more filter design cases in the database, the more chances that the user can find a suitable fit.

Filter design tools such as FILTERCAD and SCHEMATICA are used in either modifying an existing design or in helping to create new design. This has however only been demonstrated manually at this stage due to the incompatibility between these external tools and the developed application using Visual Basic. Advanced code development is needed to resolve this to ensure the automation of this feature in any future system, such as the full working (commercial) version.

Using the expert system, it is clear that it is designed to be used to complement the user's design abilities by providing the system help that is needed during the design. Thus the system is not a substitute for the user, but the two are working together as a part of the system intelligence. The user will benefit from the computer system in terms of speed, storage and access to information. The system was designed specifically to ease the communication with the user through an appropriate interface.

As stated in the thesis, the main objective in the project was to develop an expert system to support electronic design within a mechatronic system and hence to help users at the top-level by supporting the transfer of design requirements to the lower level systems. As discussed in Chapter 4, there were several methods of representing knowledge in an AI system but after the knowledge acquired from the experts and other sources for filter design was gathered and analysed, two representation techniques were chosen, production rules and frames. These enable the efficient capturing of filter design knowledge and produce a structure that supports effective filter design problem solving. As the filter design knowledge was analysed, that led to extending the frame to use a unified filter design template to capture different filter designs. On increasing the number of design cases there was a need to document these cases in a database. These filter design cases were stored conveniently using

Microsoft Access. The system is then able to retrieve data from the database as well as store new data.

The advantage in using a database over files was that some applications incorporate data such as filter circuit diagrams and images, which can be handled easily using the database. The other advantages were from the implementation perspective where a consistency in the programming language used and the database. Microsoft Visual Basic was used to design the user front end, which can be linked easily to the Microsoft Access database and to Structured Query Language (SQL). The use of a database therefore eases the design and makes the implementation more efficient.

The use of a conventional rule-based expert system to implement the proposed system was felt to risk isolating the user from being part of the design process. That resulted a conflict with the main objective of designing and implementing an expert system that forms a complementary part in the design process and associated making decision. It was found that rules are useful in representing the human expert knowledge such as the system help and in following the design procedure, but not practical when used for searching and modifying designs held in the database.

The research led to the use, in additional to rules, of a Case Based Reasoning approach, based on design reuse and case adaptation. An integration of rule-based and case based reasoning was therefore used where the case base is represented by the design cases stored in the database. This integration allowed other sources of knowledge to be used more effectively in forming the design cases while supporting the capture of expertise through the creation of new designs and setting the system help.

Although many commercial CBR development tools are now available, none was used in proving that a system is achievable, irrespectively of what implementation tools are used. The Visual Basic programming language has been chosen to implement the CBR expert system, with the case base stored in a Microsoft Access database. As the user here is tightly involved in the process, the user interface of the system has to be designed from the user's perspective, for which Visual Basic is well suited. The SQL language was also used for searching the cases in the database and this is configured in Visual Basic.

The system interface was designed by following the standard windows approach for screen design. The aim for the user interface was to accommodate users who are both experienced and novices in filter design. The layout of the filter design expert system was based on forms, which allow users to fill in the blanks and select options with the mouse when working with the system. The forms as designed included title and field headings, provide default values when practical, and display data at an appropriate field length.

The system was also designed to provide appropriate feedback, which will make the user's interaction more enjoyable. Not providing feedback is a certain way to frustrate and confuse. Feedback was involved in status information, error or warning messages and so forth.

The help system was designed so as to provide both general and specific information in a form in which both experienced and novice users can have access to it. The help system was based on a text filter for design knowledge which provides the filter definitions, terminology, as well as filter design methods. Additional text, graphs and diagrams of filters supported this help system.

After the system implementation was completed, it was tested to check the usability of the system. The participants for the usability test were chosen from a range of backgrounds to represent the variety of actual users. Initially, three participants were chosen to conduct the test. Each had different experience and background in filter design. They were handed the same design scenarios and every participant was given the same questionnaire. The user's comments were analysed and reviewed and the system was modified accordingly.

The usability test is an iterative process, which can take place at early stage when developing the look and feel of the prototype. As a result of this simple initial test, most of the changes to the system were in the user interface, for instance, a print command was added, and more online help was needed. In the search form that comes after the design form for instance, Participant 1 suggested that before setting the design criteria and starting a search, a summary of the filter specification should be displayed on the form, rather than having to go back to look to the previous form. The comments from the participants about the system context and functional were fairly acceptable in that all were able to follow the system and perform the tasks that they were asked to do.

7.3 Linking the Designed Electronic System to Other Domains Within Mechatronics

Having completed the implementation of filter design expert system then, as discussed in Chapter 6, the same techniques can be used to develop similar systems for the other electronic sub-systems, such as amplifiers, microprocessors, DSPs and microcontrollers. For instance, the expert system for amplifier design would be developed, as for the filters, by first contacting amplifier design experts to establish the necessary design knowledge. From this information the hierarchy tree for amplifiers, which, unlike filters, exist only for analogue signals, can be created. The knowledge structures together with amplifier design cases gained from experts as well as other sources can be represented and coded and the cases stored in the database. As with filters, the system will have links to appropriate design

and simulation tools to aid users in modifying or developing a design. Thus there are a lot of similarities between active filter design and amplifier design, which will support system development.

The other electronic components such microcontrollers and microprocessors, contain both software and hardware elements and are used as implementation technologies for sub-systems such as digital filters. As these devices are programmable, the emphasis there will be on the choice of hardware implementation and the development of its software counterpart. The hardware implementation would be based in a similar approach that was used for filter and amplifier expert system. Whereas for the software implementation would be different because of the nature of software. For instance, it might represent the required code in a saved text file, which can be called and executed by the system. Further research is needed here to identify a means by which the Case Based expert system can be used to represent a software implementation based on code reuse.

In Chapter 6, ways have been defined to link together the individual modules and sub-modules resulting from the decomposing of the mechatronics system to the different technologies. As the main interest in the thesis was the electronic domain, the focus was on linking the electronic sub-systems. With the aid of appropriate expert system strategies, these links can be achieved by setting constraints on each sub-system, which in turn will be related to the other sub-systems when undertaking a design. The Data Dictionary used to define the interconnections between sub-systems and as a means of constraint transfer between processes that represent the sub-systems. The flows of the Data Dictionary represented the links and the interfaces between the different sub-system processes functions.

Referring to the design example of water tank in Chapter 6, it is noticed that the input and output flow Data Dictionary of processes under design are updated automatically by the *DES* each time new design information appears. Each of these processes represents an electronic sub-system and its associated expert system such as that for a filter, amplifier, etc. Considering constraints on, to and from each process, it has been found that if the processes are interfaced appropriately then the system will proceed intelligently, or if there is an inconsistency between processes the system will then prompt the designer through a message identifying the conflict and its remedy. Whenever needed, the designer will have access to any of the expert systems and can feed some information from the *DES* into this and back to the *DES*, which will update accordingly.

From the design example, it has been concluded that in a top-down hierarchy tree of a complex system consisting of many sub-systems at different levels, that any change that appears in the data dictionary to any of these sub-system will propagate to other sub-

systems, both horizontally and vertically. The *DES* continues propagating the changes to all other sub-systems and updating the associated Data Dictionary flows accordingly.

The designer relies on the support provided by the system in proceeding by letting the system do as much as it can without the interference of the designer. The designer only interferes when needed or asked by the system to choose an option or take a decision in relation to the design. There is no point in the designer taking an action in the design process that can be done automatically by the system. Therefore the system is designed to interact intelligently with the designer by considering the computer capabilities in terms of help, support, searching speed and memory size. The role of the computer system is that of complementing the strengths and skills of the human designers.

7.4 Summary of Contributions

The major contributions of this thesis are as follows:

- Functional decomposition has been successfully used in the design process of mechatronic system from top to bottom levels.
- An expert system of designing an electronic component of mechatronics system has been shown to be implementable.
- This expert system is dedicated to an electronic component of mechatronics system but establishes a methodology for any electronic system.
- Full demonstration is provided using filter design as an exemplar.
- The implemented system is usable by users who are either novice or expert in electronic design.
- The system is designed to provide help to suit user needs
- It integrates rule-based and case-based approaches in the design of the expert system and develops an implementable database by using a unified template for electronic design cases.
- The impact of using fuzzy logic in setting the optimisation design criteria is evaluated, though it has been implemented using crisp sets.
- It uses a filter design expert system as a vehicle for other electronic systems such as an amplifier.
- It studies and defines ways of linking the sub-system under design to other electronic domains and to the overall mechatronics system.

7.5 Future Work

Having considered the work undertaken in this project, and the progress that has been made towards the development and implementation of the expert system of electronic system within mechatronics, it is now necessary to look forward to use the experience and understanding gained during the course of the project to highlight areas which still require consideration, and to suggest the ways in which the project should now progress.

When users used the system as implemented in Chapter 5 to find a design fit and were not fully satisfied with the results of the search then, by clicking on the '*Modify*' button, filter design tools links are launched in which the user was able to fill in design parameters manually. The user was then required to enter the results into the database. Further work is needed to automate this data transfer. Such work may require a skilled analyst and programmer to develop the implementation of the system.

The implementation of this expert system prototype is used as a vehicle to implement not only other filter types such as passive and digital but also other electronic systems. For instance, additional work can be done to design and implement an expert system for amplifier design and other electronic systems. Also for the electronic sub-systems that have a software implementation in additional to their hardware implementation such as a microcontroller, there is a need to develop and define the means in which their hardware and software designs can be represented by an expert system. This might be based on the use of the object-oriented methodology and code reuse.

Having developed the expert system for different electronic components in the mechatronics environment, and of ways of linking the electronic system under design to other electronic and non-electronic domains in the mechatronics environment as defined in this thesis then there is a need for further work to define and establish whether such a system can automate this process.

Such work may start off by developing the coding and implementation of the links in the electronic domains and then extend this to involve the other components of mechatronics and establish an impact on the overall system. This will lead to other work, which could be shared with other Universities, in identifying ways in which such a link could be implemented, such as to the Schemebuilder package developed in part by Prof. Bradley when at Lancaster University and which helps users in choosing between design alternatives in a control systems or hydraulic systems. A team would therefore have to be formed, which could undertake the different aspects of such work. This might involve in the team a mechanical engineering researcher, an electronic engineering researcher and software

engineer or programmer. This in turn requires funding to be sought to finance the continuation of the research.

8. Bibliography

Aklthoff, K. and Web, S. (1992). Case-based reasoning and expert system development. *Contemporary-Knowledge-Engineering-and-Cognition.-First-Joint-Workshop-Proceedings,* 1992: 146-58: Springer-Verlag, Berlin, Germany

Alpaydin, G. et al (2000). Multi-level optimisation approach to switched capacitor filter synthesis. *IEE Proc. Circuit devices Syst., Vol 147, No 4.*

Amerongen, J. van. et al. eds. (2002). *Proceedings of the 8th Mechatronics Forum International Conference Mechatronics 2002*, the Netherlands: Drebbel Institute for Mechatronics.

Analog Devices website, http://www.analog.com/ (accessed May 2001). Semiconductor company specializing in high-performance analogue, mixed-signal and DSP.

Awad, E. M. (1996). *Building expert systems: principles, procedures, and applications.* USA: West Publishing Company.

Babu, V. R, Mazhari, B and Hasan, M. M. (1996). An Expert System Approach to analog Circuit Synthesis. *10th International Conference on VLSI Design.* pp425-428. IEEE Comput. Soc. Press, Los Alamitos, CA, USA

Beckman, T. J. (1991). Selecting Expert System Applications. *AI Expert*, Feb., pp. 42-48.

Bishop, R. H. ed. (2002). *The Mechatronics Handbook.* USA: CRC Press LLC.

Bolton, W. (1995). *Mechatronics: electronic control systems in mechanical engineering.* UK: Addison Wesley Longman Limited.

Bradley D., et al. (2000). *Mechatronics and the design of intelligent machines and systems.* UK: Stanley Thornes.

Bradley, D. and Dorey, A. P. (1994). Measurement science and technology- essential fundamentals of mechatronics (review article), *Measurement Science and Technology* (5), pp.1415-1428

Britton C, oake J. (2000). *Object-Oriented System Development.* UK: McGraw-Hill Publishing Company.

Brugge, B. and Dutoit, A. (2000). *Object-Oriented Software Engineering: Conquering Complex & Changing Systems.* Englewood Cliffs, N. J: Prentice Hall.

Chen,W. K. (2000). *The VLSI Handbook.* USA: CRC Press LCC.

Cross, N. (1994). *Engineering Design Methods: Strategies for Product Design.* J W & Sons.

Dan Necsulescu, 2002. *Mechatronics*, Prentice hall, USA.

Darlington, K. (2000). *The Essence of Expert Systems*. UK: Pearson Education Limited.

Davies, R. (1992). Electronic System Design- Tools and Methodology to Meet the Productivity Challenge.*IEEE,* pp599- 600.

DeMarco, T. (1979). *Structured Analysis and System Specification,* Englewood Cliffs, NJ: prentice-Hall.

Dennis a., Wixom B.H. (2000), *System Analysis and design*, John Wiley & Sons, USA.

Dumas & Redish., 1999. *A practical Guide to Usability Testing*. Intellect Ltd.

Durkin, John. (1994). *Expert Systems Design and Development*. USA: Macmillan Publishing Company.

Dym, C. L. and Little, P. (2000). *Engineering Design: A Project-Based Introduction*. John Wiley & Sons.

El-Turkey, F. and Perry, E. (1989). BLADES: An Artificial Intelligence Approach to Analog Circuit Design. *IEEE Transaction on Computer aided Design,* Vol. 8, No. 6.

FilterCAD design tool. Linear Technology Inc.

Francioni, J M and Kandel, A (1988). A software engineering tool for expert system design. *IEEE Expert*, vol. 3, no. 1, pp. 33-41, 1988

Fraser C. and Milne, J. (1994). *Integrated electrical and electronic engineering for mechanical engineers*. UK: McGraw-Hill international

French M. J. (1985). *Conceptual Design for Engineer.* 2nd edition. UK: Design Council Publications.

French, Michael J., 1990. Function Costing: A potential Aid to Designer. *Journal of Engineering Design*, Vol. 1, No. 1

Gane, C. and Sarson T. (1979). *Structured System Analysis*. Englewood Cliffs, NJ: Prentice-Hall.

Giarratano J. and Riley, G. (1998). *Expert Systems Principles and Programming*. USA: PWS Publishing Company.

Greenwell, M. (1988). *Knowledge engineering for expert systems*, Ellis Horwood: Wiley.

Gustard, N. C. and Massara, R. E. (1994). On the Optimal design of Switched-Capacitor Circuits for analog and mixed-Signal Integrated Circuit Realization. *Analog Integrated Circuits and Signal Processing*. Nov.; 6(3): pp. 219-229, Kluwer Academic Publishers, Boston.

Hart, A. (1989). *Knowledge acquisition for expert systems.* 2nd edition. London: Kogan page.

Henderson, R. K., Li-Ping and Sewell, J. I. (1993). Analog integrated filter compilation. *Analog Integrated Circuits and Signal Processing.* May; 3(3): pp. 217-228. Kluwer Academic Publishers, Boston.

Herbst, L. J. (1996). *Integrated Circuit Engineering.* Oxford University Press.

Hoffer, J. A., George, J. F. and Valacich, J. S. (1999). *Modern System Analysis and Design.* USA: Addison Wesley Longman, Inc.

Hollins, B. and Pugh, S. (1990). *Successful Product Design: What to do and When.* Butterworth & Co.

Horn, M.V.1986. *Understanding Expert Systems.* USA: Bantam

http://wwwagr.informatik.uni-kl.de/~readee [Accessed March 2001]. About READEE research project to develop an intelligent reuse tool for circuit designs in electrical engineering.

JA Hoffer, J F George, J S Valacich. (1999), *Modern System Analysis and Design*, Addison Wesley Longman, Inc. USA.

Jackson, P. (1999). *Introduction to Expert Systems.* 3rd edition. USA: Addison Wesley.

Jantsch, A., Kumar, S. and Hemani, A. (2000). A metamodel for Studying Concepts in Electronic System Design. *IEEE Design & Test of Computers*, pp78-85

Jokpac, D. (1994). Electronic system design automation: from Product Idea to Gates. *Electro International.* Boston, USA, pp.663-669

Kokozinski, R., Hosticka, B. J. and Klinke, R. (1994). Rule-based adaptive configuration selection for analog design systems. *Analog-Integrated-Circuits-and-Signal-Processing.* March ; 5(2): pp.111-19, Kluwer Academic Publishers, Boston.

Kolodner J (1992). An Introduction to Case Based Reasoning. *Artificial Intelligence Review.* 6, 3-34.

Kolodner J, Simpson R and Sycara K. (1985). A Process Model of Case based Reasoning in Problem Solving. *Proc. IJCAI-85,* PP284-290. Morgan Kaufmann.

Koza J. R. et al. (2003). Evolving Inventions. *Scientific American, Inc.*

Linear Technology website, *http://www.linear-tech.com,* (accessed April 2001). Filters Home Page.

Lucent Technologies Bell Labs innovations, www.Lucent.com/minds/transistor. (accessed Feb. 2003).

Margo, V. and Etter, D. M. (1991). An Expert system for Digital Filter Design. *IEEE Proceedings of the 23rd Midwest Symptom on Circuits and Syatems.*

Matlab fuzzy logic toolbox. (2002). The Mathworks, Inc.

Matlab signal processing toolbox. (2002). The Mathworks, Inc.

Microsoft Word 2000 software.

Moore, G. (1965). 'Cramming more components onto integrated circuits', *Electronics*, Vol.38, N.8

Mostov, K and Soloviev, A. (1995). Conceptual Design of Complex Electronic System. *IEEE,* pp1837-1842

Multisim design tool.

Nie, G. L, Mahoudeaux, P. M. and Gaillard, P. (1991). Application of the AI techniques to signal processing: a knowledge-based system for digital filters synthesis. *Signal Processing,* May; 23(2): pp.121-136.

Oehler, P. et al. (1998). READEE - Decision Support for IP Selection using a Knowledge-Based Approach. *IP98 Europe Proceedings*. Miller Freeman.

Oehler, P., Grimm, C. and Waldschmidt, K. (1995). KANDIS – A Tool for Construction of Mixed Analog/Digital Systems. *Eur Des Autom Conf PROC, IEEE*, LOS ALAMITOS, CA, (USA), pp. 14-19.

Riesbeck C and Schank R (1989). *Inside Case-Based Reasoning.* Lawrence Erlbaum Associates, Inc., Publishers.

Riesgo, T. et al. (1996). The Use of Standards in Electronic Design. *IEEE*, pp407-412

Rohrer, R. A. (1988). Evolution of the Electronic Design Automation Industry. *IEEE Design & Test of Computers,* pp8-13

Rumbaugh, J., et al. (1991), *Object-Oriented Modelling and Design.* Englewood Cliffs, N.J: Prenetic-Hall.

Samardar, M. (1991). EXACT: EXpert in Analog Circuit Topology. *Expert Systems with Applications,* Vol. 2, pp. 137-152, Pergamon press Plc.

Schaaf M. et al (2002). Supporting Electronic Design Reuse by Integrating Quality Criteria into CBR Based IP Selection. *6th European Conference- Berlin.* Pp628-41

Schank R (1982). *Dynamic Memory: A Theory of Learning in Computers and People.* Cambridge, UK: Cambridge University press.

Schank, R and Slade, S. (1991). The Future of Artificial Intelligence: Learning From Experience. *Applied Artificial Intelligence.* 5(1) 97-107.
Schematica Software website, *http://www.schematica.com,* (accessed April 2001). Filter Wiz active filter designer.

Shepherd, P. (1996). *Integrated circuit design fabrication and test.* 1st edition. UK: Macmillan Press Ltd.

Shetty, D. and Kolk, R. A. (1997). *Mechatronics system design.* USA: PWS publishing.

Sheu, B. J., Fung, A. H. and Lai, Y. N. (1988). A knowledge-based approach to analog IC design. *IEEE-Transactions-on-Circuits-and-Systems.* Feb. ; 35(2): 256-8

Siu, K. W. and Lee, Y. S (1995). MATSPICE- Putting Circuit Simulation and Design Optimization Together.*ECCTD'95 European Conference on Circuit Theory & Design,* pp1161-1164, vol.2. Turkey: ITU

Smith, M. S. (1997). *Application Specific Integrated Circuit.* USA: Addison- Wesley.

Smith, P. (1996). *An introduction to knowledge engineering.* International Thomson Computer P.

Storey, N. (1998). *Electronics a system approach.* 2nd edition. UK: Addison Wesley Longman Ltd.

Sycara, K. et al (1992). CADET: A case-Based Synthesis Tool for Engineering Design. *Industrial Journal of expert System.* pp157-188

Temes, G. C. and Mitra, S. K. (1973). Modern filter theory and design. *The IEEE computer society,* Wiley

Turban, E. (1992). *Expert systems and applied artificial intelligence.* New York: Macmillan.

Turban, E. (1992). *Expert Systems and Applied Artificial Intelligence.* USA: Macmillan.

Ulrich, K. T. and Eppinger, S. D. (1995). *Product Design and Development.* McGraw-Hill.

Vincentelli, A. S. (1991). Towards Automatic Synthesis and verification of Complex Electronic systems. *IEEE,* pp888-893

Vollrath I.(1998). Reuse of Complex Electronic designs Requirements analysis for a CBR Application. *Advances in Case Based Reasoning. 4th European Workshop, EWCBR-98.* pp136-147

Wallace, W. F., Dlay, S. S. and Hinton, O. R. (1998). An expert system approach to mixed mode design partitioning. *IEE-Colloquium-on-Systems-on-a-Chip*-Ref.-No.1998/439. 1998: 3/1-5, IEE, London, UK

Walters R.M. (1996). *The high-level design of electronic systems*, PhD thesis, Lancaster University.

Walters, R.M., Bradley, D. A. and Dorey, A. P. (1999). A conceptual study for a computer-based tool to support electronics design in a mechatronic environment. *Microprocessors and Microcomputers,* 24: pp. 51-61

Wang, S. J and Lee, Y. S (1996). Developing of An Expert System for Designing Analyzing and Optimizing Power Converters. *19th Convention of Electrical and Electronic Engineer,* Israel, pp359-362.

Wang, S. J, Lee, Y. S. and Siu, K. W. (1997). Expert System Aided Design, Simulation and optimization of Power Converter. *23rd International Conference on Industrial Electronics, Control and Instrumentation*, pp1016-1021, IEEE, NewYork, USA

Wang, S. J., Siu, K. W. and Lee, Y. S. (1997). Applying expert system and fuzzy logic to power converter. *IEEE International Symposium on Circuits and Systems*. Hong Kong, pp1732-1735

Watson, Ian. (1997). *Applying case based reasoning: techniques for enterprise systems.* USA: Morgan Kaufmann Publishers, Inc.

Watson, Ian. (1997). *Applying case based reasoning: techniques for enterprise systems.* USA: Morgan Kaufmann Publishers, Inc.

Whitaker, J.C. ed. (2002). *Electronic system maintenance handbook.* 2nd edition. USA: CRC Press LLC.

William, A & Taylor, F (1995). *Electronic filter design handbook.* 3rd edition, New York, McGraw-Hill.

Wu, J. G, et al. (1990). A model-based expert system for digital system design. *IEEE Design& Test of Computers,* vol. 7, no. 6, pp. 24-41

www.bell-labs.com (accessed Jan. 2003). Bell Lab timeline.

www.developer.axis.com/products/mcm (accessed March 2003). Axis produces chips and boards for networking.

www.encyclopedia.com (accessed Jan. 2003). Invention of transistor.

www.icknowledge.com, (accessed Feb. 2003). History of the integrated circuit.

www.intel.com. (accessed Feb. 2003). A history of the microprocessor.

www.kip.uni-heidelberg.de/atlas/DATA/publications/DISS/node84.html (accessed March 2003). Kirchhoff-Institut für Physik.

www.nobel.se/physics/laureates/1956 (accessed Jan. 2003). The Nobel Prize website.

www.nobel.se/physics/laureates/2000/index.html (accessed March 2003). The Nobel Prize website.

www.obducat.com/index.asp (accessed March 2003). Etching technology/multi chip module.

www.ti.com. (accessed Feb. 2003). History of innovation

Yourdon E. (1989), *Modern Structured Analysis,* Englewood Cliffs, NJ:Prentice-Hall, USA

Yourdon, E. (1989). *Modern structure method.* Prentice-Hall, Englewood Cliffs, NJ.

Yourdon, E. and Constantine L.L. (1979). *Structured Design.* Englewood Cliffs, NJ: Prentice-Hall.

Appendix A

System Modelling and Implementation

A1. The Context Diagram

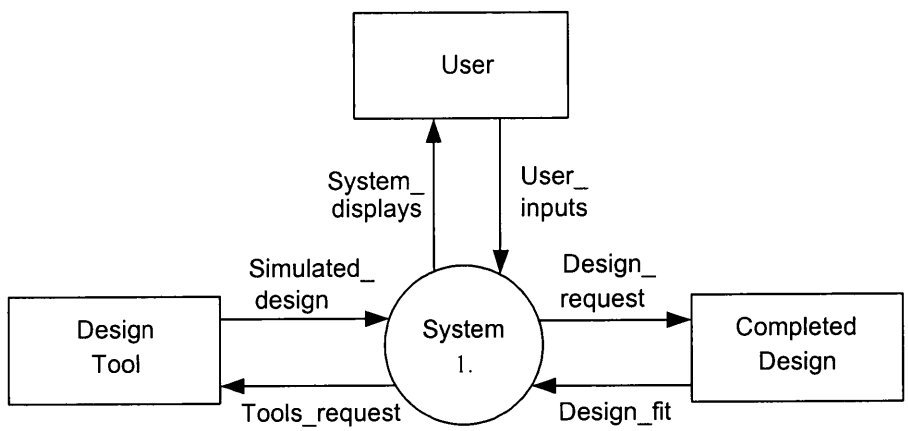


Figure A1.1: *Context diagram for filter design system*

A2. The Full Functional Decomposition

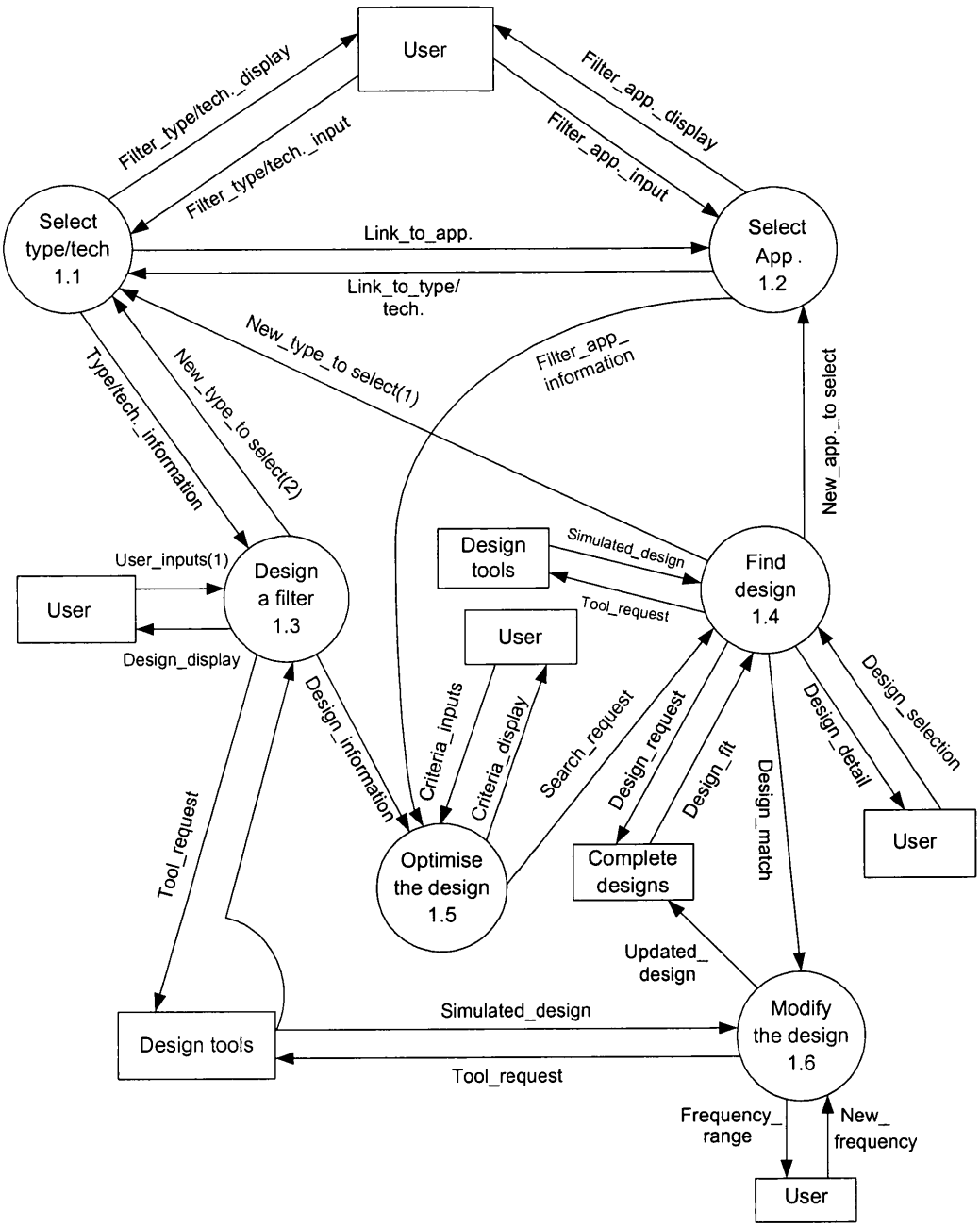


Figure A2.1: *First Level Decomposition of the Filter Design System*

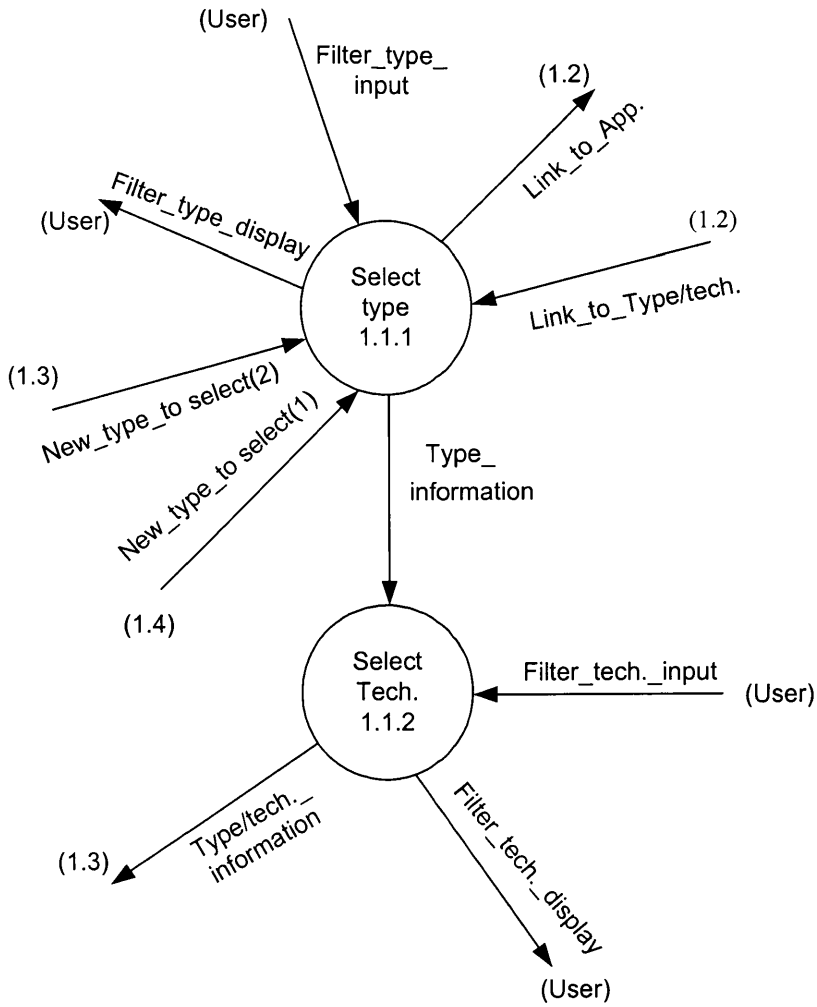


Figure A2.2: *The decomposition of process 1.1, Select Type/Tech*

Process 1.1.1: Select Type

Structured English Description:

If Filter_type_display = ON THEN Filter_type_input selected
While Filter_type_input selected DO
GOTO process 1.1.2 " Select tech."
END DO
If new_type_to select(1) = ON OR New_type_to select(2) = ON
OR Link_to_type/tech = ON THEN show filter_type_display
Else
If Link_to_app. = ON THEN GOTO Select App. Process 1.2
END

Process 1.1.2: Select Tech.

Structured English Description:

While type_information = ON DO
Filter_tech_display = ON
END DO
If Filter_tech_display = ON THEN Select Filter_tech_input
If Filter_tech_input selected THEN GOTO design a filter process 1.3
END

Process 1.2: Select Application

Structured English Description:

If filter_App_display = ON AND Filter_ app_input selected
THEN filter_app_information = ON
If New_app_to select = ON OR Link_to_app = ON
THEN Filter_app_display = ON
Else GOTO Link_to_type/tech
END

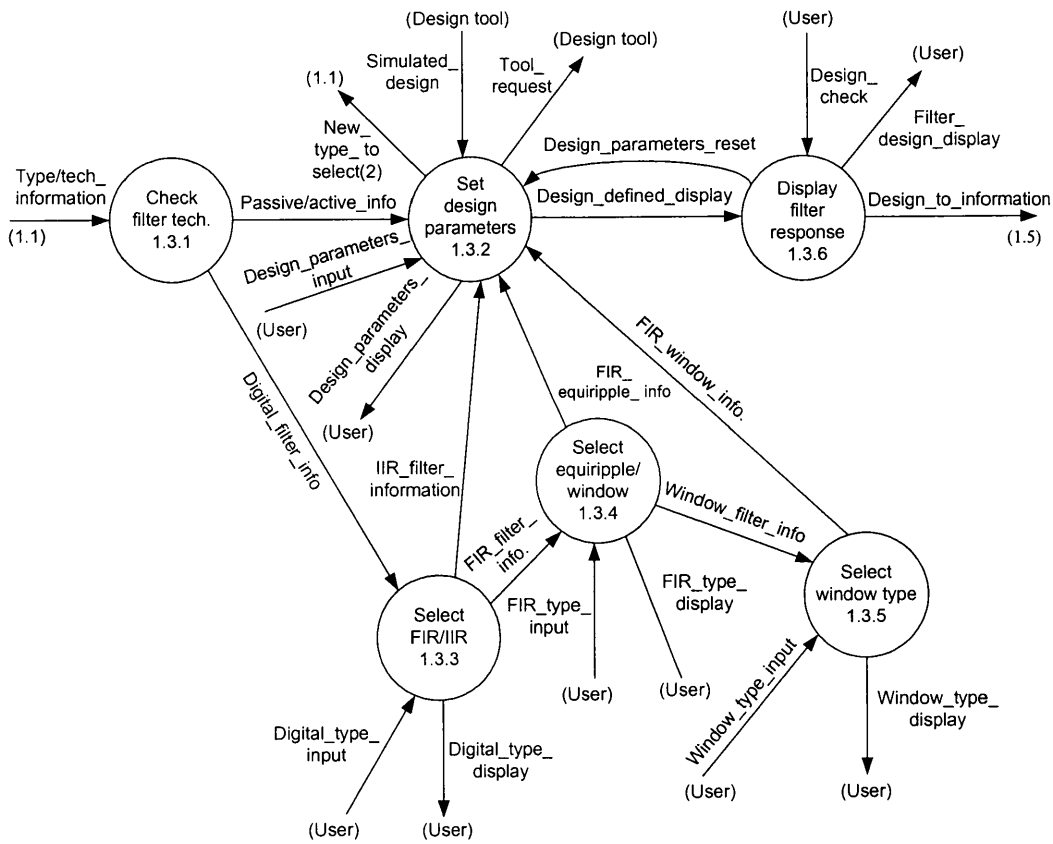


Figure A2.3: *Decomposition of process 1.3: Design a filter*

Process 1.3.1: Check Filter Tech.

Structured English Description:

If type/tech_information = passive/active_info THEN

GOTO set design parameters process 1.3.2

Else

GOTO select FIR/IIR process 1.3.3

END

Process 1.3.2: Set design parameters

Structured English Description:

IF Passive/active_info = ON OR IIR_filter_information = ON

OR FIR_equiripple_info = ON OR FIR_window_info = ON

THEN show design_parameters_display

IF design_parameters_input is selected THEN Tool_request = ON

ELSE

GOTO select type/tech process 1.1

If Simulated_design = ON AND design_parameters_rest = OFF THEN

Design_defined_display = ON

GOTO display filter response process 1.3.6

END

Process 1.3.3: Select FIR/IIR

Structured English Description:

IF Digital_filter_info = ON THEN Digital_type_display = ON

IF Digital_type_input = IIR THEN

GOTO set design parameters process 1.3.2

ELSE

IF Digital_type_input = FIR THEN GOTO select equiripple window process 1.3.4

Process 1.3.4 : Select equirriple/window

Structured English Description:

If FIR_filter_info. = ON THEN show FIR_type_display

If FIR_type_input = equiripple THEN GOTO process 1.3.2 "Set design parameters"

ELSE

If FIR_type_input = window THEN GOTO process 1.3.5 " Select window type"

END

Process 1.3.5 : Select window type

Structured English Description:

IF Window_filter_info = ON THEN show window_type_display

IF window_type_input = Kaiser OR

window_type_input = Hann OR

window_type_input = Hamming OR

window_type_input = Triangule OR

window_type_input = Chebyshev OR

window_type_input = Blackmann THEN

GOTO process 1.3.2 " Set design parameters"

END

Process 1.3.6 : Display filter response

Structured English Description:

If design_defined_display = ON THEN Filter_design_display = ON

If design_check = ON AND Design_parameter_reset = OFF

THEN GOTO process 1.5 " Optimise the design"

END

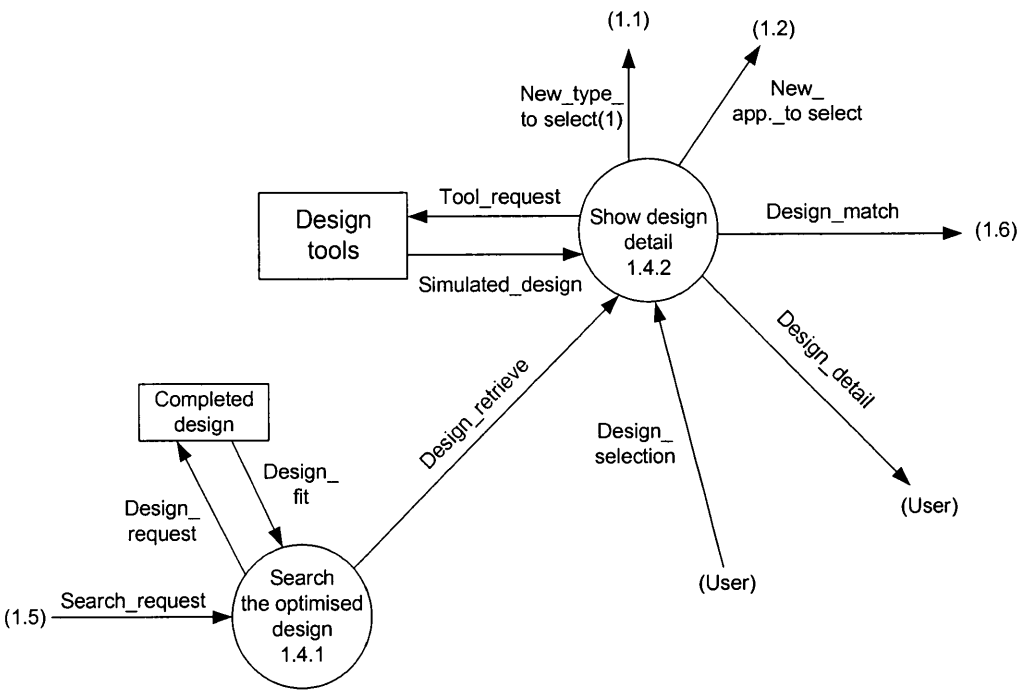


Figure A2.4: *Decomposition of process 1.4: Find design*

Process 1.4.1: Search the optimised design

Structured English Description:

IF Search_request = ON AND Design_request = ON

THEN Design_fit = ON

Design_return = ON

END

Process 1.4.2 : Show design detail

Structured English Description:

If design_return = ON AND Design_selection = ON

THEN Design_detail is displayed

ELSE

IF Design_return = OFF

THEN New_type_to select(1) OR New_app_to select = ON

If Tool_request = ON

THEN Simulated_design = ON

END

Process 1.5 : Optimise the design

Structured English Description:

If design_information OR Filter_app_information = ON

THEN Criteria_display = ON

IF Criteria_display = ON

THEN Criteria_inputs = ON

IF Criteria_inputs = ON

THEN Search_request + ON

GOTO process 1.4 " Find design"

END

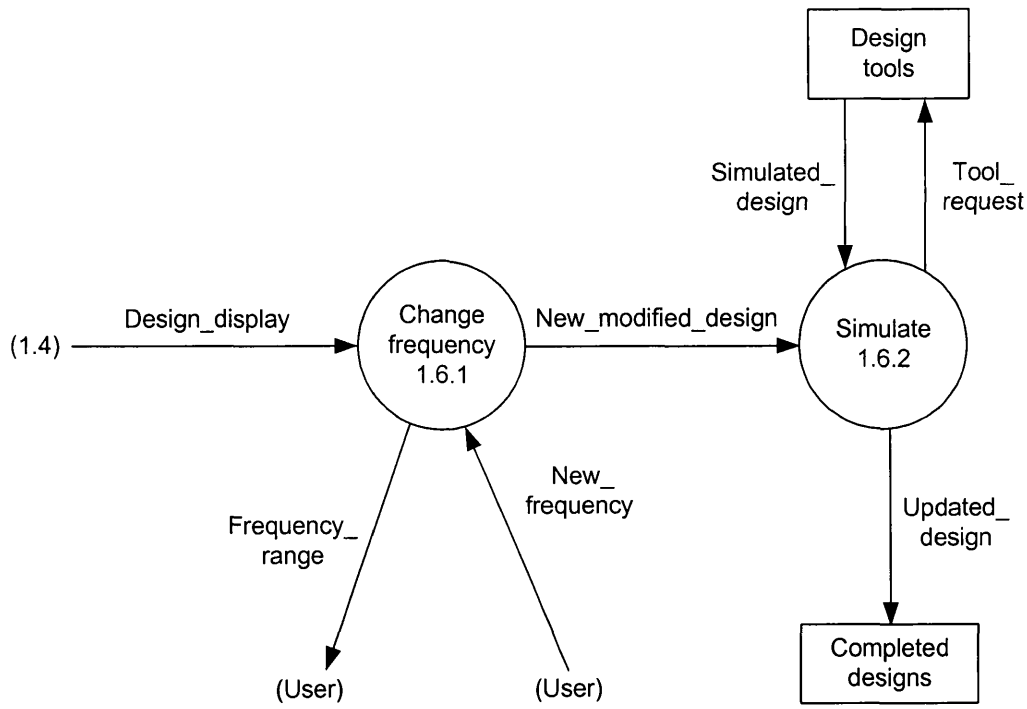


Figure A2.5: *Decomposition of process 1.4: Modify the design*

Process 1.6.1: Change frequency

Structured English Description:

If Design_display = ON

THEN Frequency_range = ON

If New_frequency_input selected

THEN New_modified_design = ON

END

Process 1.6.2: Simulation

Structured English Description:

IF New_modified_design AND Tool_request = ON

THEN Simulated_design = ON

IF Simulated_design = ON

THEN Updated_design = ON

GOTO Completed designs

END

A3. The Data Dictionary

Data Dictionary Entry

Name: user_Inputs

Flows Included: filter_type/tech._input; filter_type_input; filter_tech._input; filter_app._input; user_inputs(1); design_selection; criteria_inputs; new_frequency input.

Data Dictionary Entry

Name: system_Displays

Flows Included: filter_type/tech._display; filter_app._display; filter_type_display; filter_tech._display; design_display; design_list; frequency_range; criteria_display.

Data Dictionary Entry

Name: user_Inputs(1)

Flows Included: design_parameters_input; digital_type_input; FIR_type_input; window_type_input; design_check.

Data Dictionary Entry

Name: Design_display

Flows included: design_parameters_display; digital_type_display; FIR_type_display; window_type_display; filter_design_display.

Data Dictionary entry

Name: filter_type/tech_input

Form: text data

Sets the filter type and technology by the user

Data Dictionary entry

Name: filter_type/tech_display

Form: text data

To display information of filter type and technology

Data Dictionary entry

Name: new_modified_design

Form: text data

Data Dictionary entry

Name: link_to_app.

Form: command button

Links the flow back to application process 1.2

Data Dictionary entry

Name: link_to_type/tech

Form: command button

Links the flow back to type/tech process 1.1

Data Dictionary entry

Name: filter_type_input

Form: text data

Sets the required filter type such as lowpass, highpass etc.

Data Dictionary Entry

Name: filter_app._inputs

Form: text data

Flows included: filter application input selected; search /design input

Data Dictionary Entry

Name: filter_tech._input

Form: text data

Sets the required filter technology such as passive, active and digital

Data Dictionary Entry

Name: filter_app._ information

Form: text

Define the filter application that selected

Data Dictionary Entry

Name: criteria_inputs

Form: text list

Sets the search to the required optimising criteria as cost, implementation etc

Data Dictionary Entry

Name: criteria_display

Form: text list

Displays the optimising design criteria as cost, implementation where it can be set by users

Data Dictionary Entry

Name: new_frequency

Form: text data

Changes the searched result to the user's required frequency

Data Dictionary Entry

Name: filter_type_display

Form: text list

Display the filter type list such as lowpass, highpass etc

Data Dictionary Entry

Name: filter_tech._display

Form: text list

Displays the filter technology list such as passive, active and digital.

Data Dictionary Entry

Name: filter_app._display

Form: text list

Displays filter applications list.

Data Dictionary Entry

Name: design_list

Form: a text data

Displays the searched results include the return fit, where the user can select details, modify or reject.

Data Dictionary Entry

Name: frequency_range

Form: a text data

Displays the frequency range, where users can set the new frequency

Data Dictionary Entry

Name: completed designs

Form: database file

Represents the existing filter design cases as well the modified cases can be stored

Data Dictionary Entry

Name: design_parameters_input

Form: text data input

Sets the initial design to the user requirements such as cutoff frequency, gain etc.

Data Dictionary Entry

Name: digital_type_input

Form: text data input

Sets the digital type design to FIR or IIR

Data Dictionary Entry

Name: FIR_type_input

Form: text data input

Sets the FIR digital type to be equiripple or window type

Data Dictionary Entry

Name: window_type_input

Form: text data input

Sets the window FIR type to user's required window type

Data Dictionary Entry

Name: design_check

Form: command buttons input

Represents the user's satisfaction of the initial design to go ahead or reset design parameters

Data Dictionary Entry

Name: design_parameters_display

Form: text list

Displays the input parameters where users can set them

Data Dictionary Entry

Name: digital_type_display

Form: text input

Sets the digital filter type to the required one, either FIR or IIR

Data Dictionary Entry

Name: FIR_type_display

Form: text input

Sets the FIR type to equiripple or window

Data Dictionary Entry

Name: window_type_display

Form: text list

Sets the window type to the required one

Data Dictionary Entry

Name: filter_design_display

Form: a command button

Display the initial filter design details to the user

Data Dictionary Entry

Name: simulated_design

Form: data file

Represents the filter design in visual display where users can simulate the design

Data Dictionary Entry

Name: tool_request

Form: a command button

Links the system to external design applications

Data Dictionary Entry

Name: type/tech._information

Form: text data

Holds information of lowpass, highpass, active or digital etc. then goes to design process to set the design parameters

Data Dictionary Entry

Name: new_type_to select(1)

Form: data link

Links the user back to reselect new filter type

Data Dictionary Entry

Name: new_type_to select(2)

Form: data link

Links the user back to reselect new filter type

Data Dictionary Entry

Name: new_app._to select

Form: data link

Links the user back to reselect new application

Data Dictionary Entry

Name: type_information

Form: text data

Holds information of filter type such as lowpass, highpass etc.

Data Dictionary Entry

Name: design_information

Form: data file

Links to the optimisation process to set criteria

Data Dictionary Entry

Name: search_request

Form: data file

Links to the database file to search

Data Dictionary Entry

Name: design_match

Form: data file

Displays the searched result of filter design, which links to modify process or stored cases

Data Dictionary Entry

Name: updated_design

Form: data file

Contains the new modified designs and to be stored as a new designs

Data Dictionary Entry

Name: digital_filter_info.

Form: text data

Links to process where FIR and IIR type can be selected

Data Dictionary Entry

Name: passive/active_info.

Form: text data

Goes to set parameters

Data Dictionary Entry

Name: IIR_filter_info.

Form: text data

It goes to set parameters

Data Dictionary Entry

Name: FIR_filter_info.

Form: text data

It goes to FIR type process to select equiripple or window

Data Dictionary Entry

Name: window_filter_info.

Text data

It goes to window type process to select window type

Data Dictionary Entry

Name: FIR_equiripple_info.

Form: text data

Links to set parameters process

Data Dictionary Entry

Name: FIR_window_info.

Form: text data

Links to set parameters process

Data Dictionary Entry

Name: design_defined_display

Form: text data

Displays the initial design settings where the user can accept or reset

Data Dictionary Entry

Name: design_return

Form: text data

Contains the return cases from the search process by optimised search and it links to show design process

Data Dictionary Entry

Name: design_selection

Form: text data

To select the appropriate retrieved design

Data Dictionary Entry

Name: design_detail

Form: text data

Displays for the selected design

Data Dictionary Entry

Name: design_retrieve

Form: text data

Retrieves the design from the database file

Data Dictionary Entry

Name: design_request

Form: text data

To request a design from the database file

Data Dictionary Entry

Name: new_modified_design

Form: text data

It links to simulation process

4. The Coding

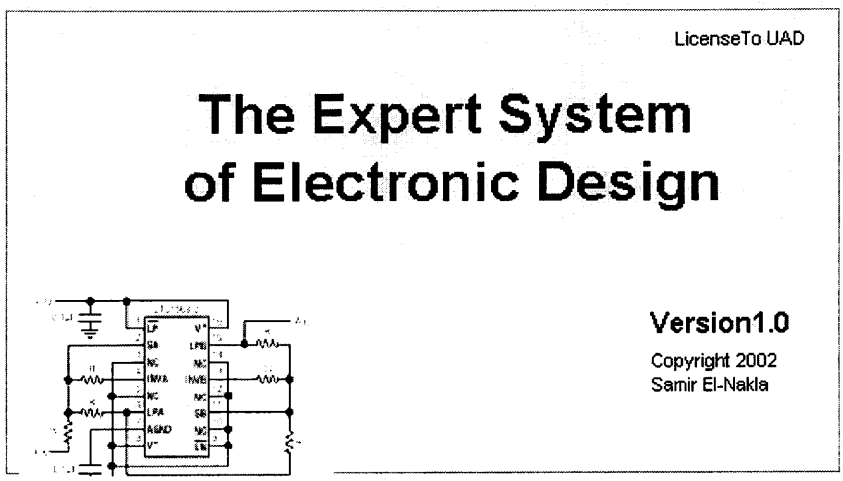


Figure A4.1: *Splash Screen for the filter design expert system*

The code for the form in Fig. A4.1:

```
Private Sub Form_KeyPress(KeyAscii As Integer)
' Go on to form1 where list of electronic systems
Unload Me
Form1.Show
End Sub


Private Sub Frame1_Click( )
' Go on to form1 where list of electronic systems
Form_KeyPress (0)
End Sub


Private Sub tmrProceed_Timer( )
' Go on to main form if no user input
Form_KeyPress (0)
End Sub
```

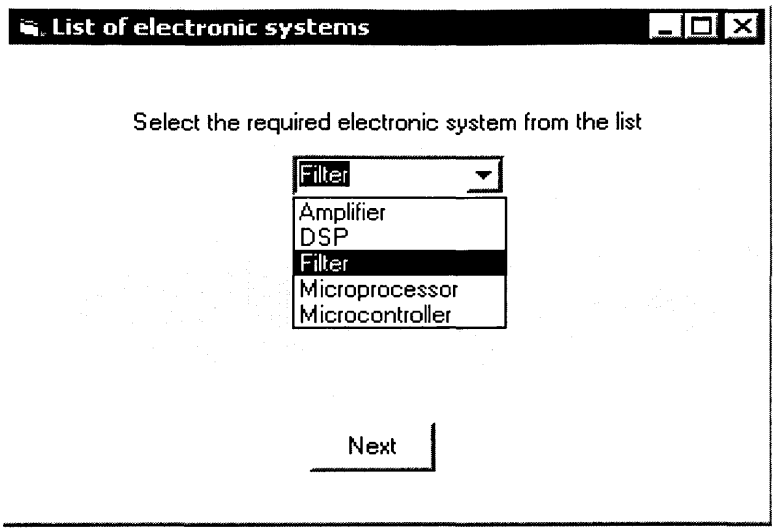


Figure A4.2: *Form of "list of electronic system"*

The code for the form in Fig. A4.2:

```
Private Sub CboList_Click( )
'display list of electronic system where filter can be chosen
If CboList.Text <> "Filter" Then
   MsgBox " For purpose of demonstration only Filter can be selected" _
   , vbInformation, "Electronic system"
End If
End Sub


Private Sub cmdNext_Click( )
' Go on to either design by application or custom design
If CboList.Text <> "Filter" Then
        MsgBox " For purpose of demonstration only Filter can be selected" _
, vbInformation, "Filter technology"
CboList.SetFocus
Else
Form2.Show

End If

End Sub
```

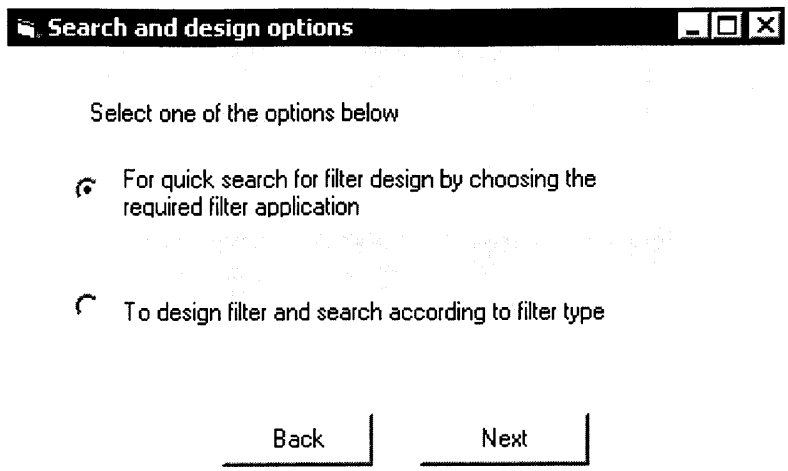


Figure A4.3: *Form of " search and design options"*

The code for the form in Fig. A4.3:

```
Private Sub Command1_Click( )
'Go to either design by application or custom design form
If optApp.Value Then
   frmApp.Show
Else
   If optDeisgn.Value Then
      frmTypeTech.Show
   End If
End If
End Sub
```

```
Private Sub Command2_Click( )
' return back to list of electronic systems form
Unload Me
End Sub
```

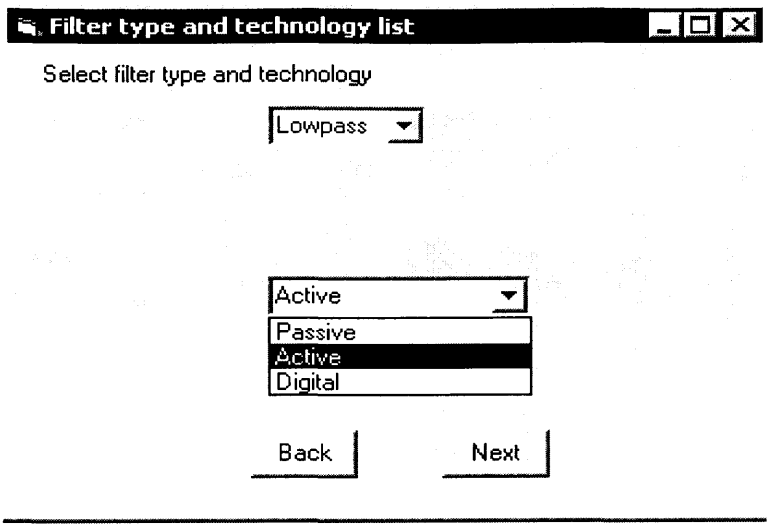


Figure A4.4: *Form of " Filter type and technology list"*

The code for the form in Fig. A4.4:

```
Private Sub cboTech_Click( )
' to ensure the selection of the Active filter for demonstarion
If cboTech.Text = "Passive" Or cboTech.Text = "Digital" Then
   MsgBox " For purpose of demonstration only Active Filter can be selected" _
   , vbInformation, "Filter technology"
End If
End Sub
```

```
Private Sub cmdNext_Click( )
' to ensure the selection of the Active filter for demonstarion
If cboTech.Text = "Passive" Or cboTech.Text = "Digital" Then
   MsgBox " For purpose of demonstration only Active Filter can be selected" _
   , vbInformation, "Filter technology"
   cboTech.SetFocus
Else
   frmdesign.Show
End If
End Sub
```

```
Private Sub Command1_Click()
'back to search and design options form
Unload Me
End Sub
```

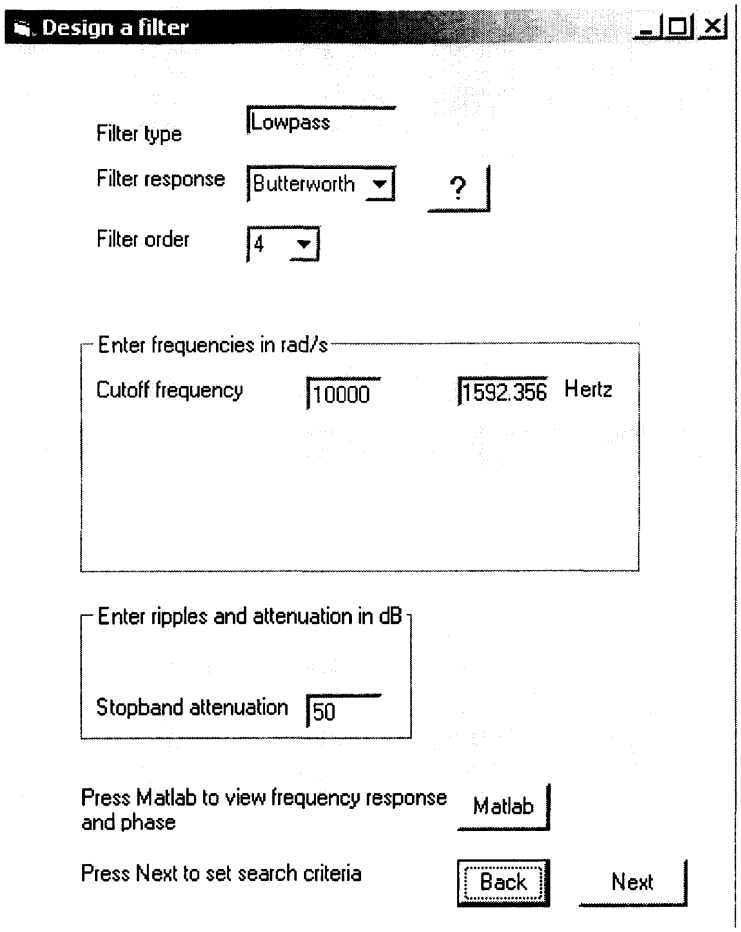


Figure A4.5: *Form of " Design a filter"*

The code for the form in Fig. A4.5:

```
Private Sub cboResponse_click( )
' set the response combo box
txtPassRipple.Visible = True
txtStopRipple.Visible = True
Label5.Visible = True
Label6.Visible = True
fraRipples.Visible = True
If cboResponse.Text = "Butterworth" Then
   txtPassRipple.Visible = False
   Label5.Visible = False
End If
If cboResponse.Text = "Chebyshev2" Then
   txtPassRipple.Visible = False
```

```
   Label5.Visible = False
End If
End Sub


Private Sub cmdMatlab_Click( )
'Lunch Matlab to display the frequency and phase response
'for the current record
'declare the filter variables
Dim Order
Dim Cutoff
Dim response
Dim ripple1
Dim ripple2
Dim filtertype
Dim upper
Dim lower
Order = cboOrder.Text
Cutoff = txtCutoff.Text
response = cboResponse.Text
ripple1 = txtPassRipple.Text
ripple2 = txtStopRipple.Text
filtertype = lblType.Caption
upper = txtUpper.Text
lower = txtLower.Text
Open "C:\Matlab-complete\work\case2.m" For Output As 1
'open file to store filter variables
Select Case filtertype '
'write filter variables into the file according to each filter type
Case "Lowpass"
If (response = "Butterworth") Then

 Print #1, "[b,a]=butter(" + Order + "," + Cutoff + ",'s')"
 Print #1, "freqs(b,a)"
 Else
   If (response = "Chebyshev1") Then
   Print #1, "[b,a]=cheby1(" + Order + "," + ripple1 + "," + Cutoff _
 + ", 's')"
 Print #1, "freqs(b,a)"
   Else
```

```
 If (response = "Chebyshev2") Then
  Print #1, "[b,a]=cheby2(" + Order + "," + ripple2 + "," + Cutoff _
+ ", 's')"
 Print #1, "freqs(b,a)"
Else
Print #1, "[b,a]=ellip(" + Order + "," + ripple1 + "," + ripple2 _
+ "," + Cutoff + ", 's')"
Print #1, "freqs(b,a)"
 End If
 End If
End If
Case "Highpass"
If (response = "Butterworth") Then
 Print #1, "[b,a]=butter(" + Order + "," + Cutoff + ",'high', 's')"
 Print #1, "freqs(b,a)"
 Else
   If (response = "Chebyshev1") Then
   Print #1, "[b,a]=cheby1(" + Order + "," + ripple1 + "," + Cutoff _
 + ",'high', 's')"
 Print #1, "freqs(b,a)"
   Else
 If (response = "Chebyshev2") Then
  Print #1, "[b,a]=cheby2(" + Order + "," + ripple2 + "," + Cutoff _
+ ",'high', 's')"
 Print #1, "freqs(b,a)"
Else
Print #1, "[b,a]=ellip(" + Order + "," + ripple1 + "," + ripple2 _
+ "," + Cutoff + ",'high', 's')"
Print #1, "freqs(b,a)"
   End If
 End If
End If
Case "Bandpass"
If (response = "Butterworth") Then
 Print #1, "[b,a]=butter(" + Order + ",[" + lower + "," + upper _
+ "], 's')"
 Print #1, "freqs(b,a)"
 Else
   If (response = "Chebyshev1") Then
   Print #1, "[b,a]=cheby1(" + Order + "," + ripple1 + ",[" + lower _
```

```
 + "," + upper + "], 's')"
 Print #1, "freqs(b,a)"
  Else
 If (response = "Chebyshev2") Then
  Print #1, "[b,a]=cheby2(" + Order + "," + ripple2 + ",[" + lower _
+ "," + upper + "], 's')"
 Print #1, "freqs(b,a)"
Else
Print #1, "[b,a]=ellip(" + Order + "," + ripple1 + "," + ripple2 _
+ ",[" + lower + "," + upper + "], 's')"
Print #1, "freqs(b,a)"
 End If
 End If
End If
Case Else
If (response = "Butterworth") Then
 Print #1, "[b,a]=butter(" + Order + ",[" + lower + "," + upper _
+ "],'stop', 's')"
 Print #1, "freqs(b,a)"
 Else
   If (response = "Chebyshev1") Then
   Print #1, "[b,a]=cheby1(" + Order + "," + ripple1 + ",[" + lower _
 + "," + upper + "],'stop', 's')"
 Print #1, "freqs(b,a)"
  Else
 If (response = "Chebyshev2") Then
  Print #1, "[b,a]=cheby2(" + Order + "," + ripple2 + ",[" + lower _
+ "," + upper + "],'stop','s')"
 Print #1, "freqs(b,a)"
Else
Print #1, "[b,a]=ellip(" + Order + "," + ripple1 + "," + ripple2 _
+ ",[" + lower + "," + upper + "],'stop', 's')"
Print #1, "freqs(b,a)"
  End If
 End If
End If
End Select
Close ' close the file
 Dim stAppName As String
 ' lunch Matlab with the Figure display the filter response
```

```
   stAppName = "C:\Matlab-complete\bin\win32\matlab.exe /r case2"
   Call Shell(stAppName, 2)
End Sub


Private Sub cmdContinue_Click( )
' this to check that all filter fields are selected
' also to check that data are vaild
If txtCutoff.Visible = True And txtCutoff.Text = "" Then
' check the cutoff frequency
   MsgBox "Please enter the Cutoff frequency"
   txtCutoff.SetFocus
Exit Sub
Else
' check the lower cutoff frequency
   If txtLower.Visible = True And txtLower.Text = "" Then
      MsgBox "Please enter the lower cutoff frequency"
      txtLower.SetFocus
   Exit Sub
   Else
   ' check the upper cutoff frequency
      If txtUpper.Visible = True And txtUpper.Text = "" Then
         MsgBox "Please enter the upper cutoff frequency"
         txtUpper.SetFocus
      Exit Sub
      End If
   End If
End If
If Val(txtLower.Text) > Val(txtUpper.Text) Then
' make sure that upper frequency is bigger than the lower one
   MsgBox "upper cutoff frequency must be greater than lower one"
   txtUpper.SetFocus
Exit Sub
End If
cmdMatlab.Visible = True
Label10.Visible = True
cmdNext.Visible = True
Label12.Visible = True
cmdContinue.Visible = False
End Sub
```

```
Private Sub cmdBack_Click( )
' back to search and options form
Unload Me
End Sub


Private Sub cmdNext_Click( )
' move the search form
frmSearch.Show
End Sub


Private Sub Form_Load( )
'on loading the form some defaults are set
lblType.Caption = frmTypeTech.CboType.Text
'filter type is selected from typeTech form
'other fields are set accordingly
txtCutoff.Visible = True
txtLower.Visible = True
txtUpper.Visible = True
Label2.Visible = True
Label3.Visible = True
Label4.Visible = True
If lblType.Caption = "Lowpass" Or lblType.Caption = "Highpass" Then
   txtLower.Visible = False
   txtUpper.Visible = False
   Label3.Visible = False
   Label4.Visible = False
   lblCutoff.Visible = True
   Label9.Visible = True
Else
   If lblType.Caption = "Bandpass" Or lblType.Caption = "Bandstop" Then
      txtCutoff.Visible = False
      Label2.Visible = False
      lblLower.Visible = True
      lblUpper.Visible = True
      Label11.Visible = True
      Label13.Visible = True
   End If
```

```
End If
txtPassRipple.Visible = True
txtStopRipple.Visible = True
Label5.Visible = True
Label6.Visible = True
fraRipples.Visible = True
If cboResponse.Text = "Butterworth" Then
txtPassRipple.Visible = False
Label5.Visible = False
End If
If cboResponse.Text = "Chebyshev2" Then
txtPassRipple.Visible = False
Label5.Visible = False
End If
End Sub

Private Sub txtCutoff_Change( )
'check the cutoff frequency limit
Dim response
If Val(txtCutoff.Text) > 100000 Then
response = MsgBox("Please enter frequency between 1 and 100000 rad/s", vbOKOnly _
+ vbExclamation, "Frequency limit")
End If
If response = vbOK Then
txtCutoff.SetFocus
txtCutoff.Text = " "
lblCutoff.Caption = ""
End If
lblCutoff.Caption = (1 / 6.28) * Val(txtCutoff.Text)
lblCutoff.Visible = True
Label9.Visible = True
End Sub

Private Sub txtLower_Change( )
' check the lower frequency limit
Dim response1
If Val(txtLower.Text) > 100000 Then
response1 = MsgBox("Please enter frequency between 1 and 100000 rad/s", vbOKOnly _
+ vbExclamation, "Frequency limit")
```

```
End If
If response1 = vbOK Then
txtLower.SetFocus
txtLower.Text = " "
lblLower.Caption = " "
End If
lblLower.Caption = (1 / 6.28) * Val(txtLower.Text)
lblLower.Visible = True
Label11.Visible = True
End Sub
```

```
Private Sub txtPassRipple_LostFocus( )
'check the range limit of passband ripple
Dim response3
If Val(txtPassRipple.Text) < 0.01 Or Val(txtPassRipple.Text) > 3 Then
response3 = MsgBox("Please enter Passband ripple between 0.01 and 3 dB", vbOKOnly _
+ vbExclamation, "Passband Ripple")
If response3 = vbOK Then
txtPassRipple.SetFocus
txtPassRipple.Text = " "
End If
End If
End Sub
```

```
Private Sub txtStopRipple_LostFocus( )
'check the range limit of stopband ripple
Dim response4
If Val(txtStopRipple.Text) < 1 Or Val(txtStopRipple.Text) > 100 Then
response4 = MsgBox("Please enter Stopband ripple between 1 and 100 dB", vbOKOnly _
+ vbExclamation, "Passband Ripple")
If response4 = vbOK Then
txtStopRipple.SetFocus
txtStopRipple.Text = " "
End If
End If
End Sub
```

```
Private Sub txtUpper_change( )
'check the upper frequency limit
Dim response2
If Val(txtUpper.Text) > 100000 Then
response2 = MsgBox("Please enter frequency between 1 and 100000 rad/s", vbOKOnly _
+ vbExclamation, "Frequency limit")
End If
If response2 = vbOK Then
txtUpper.SetFocus
txtUpper.Text = " "
lblUpper.Caption = " "
End If
lblUpper.Caption = (1 / 6.28) * Val(txtUpper.Text)
lblUpper.Visible = True
Label13.Visible = True
End Sub
```

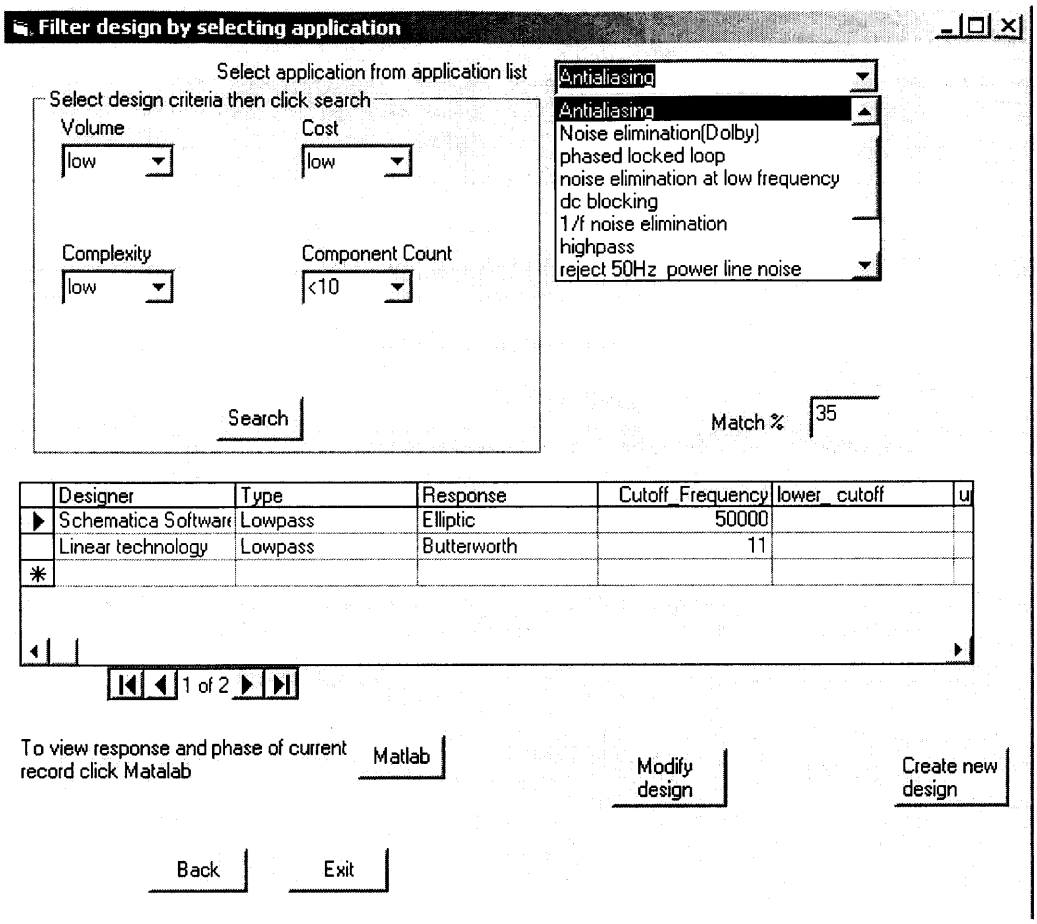


Figure A4.6: *Form of "Filter design by selecting application"*

The code for the form in Fig. A4.6:

Private Sub cboApp_Click( )

' select the filter application and the design criteria will display

Frame1.Visible = True

End Sub

Private Sub cmdBack_Click( )

' Back to Search and design options form

Unload Me

End Sub

```
Private Sub cmdExit_Click( )
Const title = "Closing the application"
Dim prompt  As String
Dim msgType As Integer
' confirm the exit request
prompt = "This will close the application."
prompt = prompt & Chr(13) & Chr(13) & "Are you sure ?"
msgType = vbCritical + vbYesNo + vbDefaultButton2
If MsgBox(prompt, msgType, title) = vbNo Then
   Cancel = True
Else
   Unload Me
   Unload Form1
   Unload Form2
End If
End Sub


Private Sub CmdSearch_Click( )
'display the retrieved record on the FlexGrid
' also to display the percentage match for the current record
DatApp.RecordSource = "SELECT * from filter_cases where Application = '" & _
            CboApp.Text & "'"
DatApp.Refresh
cmdModify.Visible = True
cmdNew.Visible = True
DBGrid1.Visible = True
DatApp.Visible = True
cmdexit.Visible = True
cmdMatlab.Visible = True
Label2.Visible = True
Label4.Visible = True
txtTotalScore.Visible = True
End Sub


Private Sub cmdMatlab_Click( )
'Lunch Matlab to display the frequency and phase response
'for the current record
```

```
Dim Order   ' declare the filter variables
Dim Cutoff
Dim response
Dim ripple1
Dim ripple2
Dim filtertype
Dim upper
Dim lower
Order = txtOrder.Text
Cutoff = txtCutoff.Text
response = txtResponse.Text
ripple1 = txtPassRipple.Text
ripple2 = txtStopRipple.Text
filtertype = txtType.Text
upper = txtUpper.Text
lower = txtLower.Text
Open "C:\Matlab-complete\work\case2.m" For Output As 1
'open file to store filter variables
Select Case filtertype '
'write filter variables into the file according to each filter type
Case "Lowpass"
If (response = "Butterworth") Then
   Print #1, "[b,a]=butter(" + Order + "," + Cutoff + ",'s')"
   Print #1, "freqs(b,a)"
 Else
   If (response = "Chebyshev1") Then
      Print #1, "[b,a]=cheby1(" + Order + "," + ripple1 + "," + Cutoff + _
     ", 's')"
     Print #1, "freqs(b,a)"
    Else
     If (response = "Chebyshev2") Then
      Print #1, "[b,a]=cheby2(" + Order + "," + ripple2 + "," + Cutoff + _
     ", 's')"
     Print #1, "freqs(b,a)"
     Else
        Print #1, "[b,a]=ellip(" + Order + "," + ripple1 + "," + ripple2 + _
        "," + Cutoff + ", 's')"
        Print #1, "freqs(b,a)"
```

```
      End If
    End If
End If
Case "Highpass"
If (response = "Butterworth") Then
    Print #1, "[b,a]=butter(" + Order + "," + Cutoff + ",'high', 's')"
   Print #1, "freqs(b,a)"
  Else
    If (response = "Chebyshev1") Then
    Print #1, "[b,a]=cheby1(" + Order + "," + ripple1 + "," + Cutoff + _
   ",'high', 's')"
   Print #1, "freqs(b,a)"
      Else
      If (response = "Chebyshev2") Then
       Print #1, "[b,a]=cheby2(" + Order + "," + ripple2 + "," + Cutoff + _
      ",'high', 's')"
      Print #1, "freqs(b,a)"
      Else
      Print #1, "[b,a]=ellip(" + Order + "," + ripple1 + "," + ripple2 + _
      "," + Cutoff + ",'high', 's')"
      Print #1, "freqs(b,a)"
       End If
   End If
End If
Case "Bandpass"
If (response = "Butterworth") Then
 Print #1, "[b,a]=butter(" + Order + ",[" + lower + "," + upper + "], 's')"
Print #1, "freqs(b,a)"
Else
   If (response = "Chebyshev1") Then
   Print #1, "[b,a]=cheby1(" + Order + "," + ripple1 + ",[" + lower + _
 "," + upper + "], 's')"
 Print #1, "freqs(b,a)"
   Else
   If (response = "Chebyshev2") Then
    Print #1, "[b,a]=cheby2(" + Order + "," + ripple2 + ",[" + lower + _
   "," + upper + "], 's')"
   Print #1, "freqs(b,a)"
   Else
```

```
    Print #1, "[b,a]=ellip(" + Order + "," + ripple1 + "," + ripple2 + _
    ",[" + lower + "," + upper + "], 's')"
    Print #1, "freqs(b,a)"
    End If
   End If
End If
Case Else
If (response = "Butterworth") Then
  Print #1, "[b,a]=butter(" + Order + ",[" + lower + "," + upper + _
 "],'stop', 's')"
 Print #1, "freqs(b,a)"
 Else
    If (response = "Chebyshev1") Then
    Print #1, "[b,a]=cheby1(" + Order + "," + ripple1 + ",[" + lower + _
  "," + upper + "],'stop', 's')"
  Print #1, "freqs(b,a)"
    Else
    If (response = "Chebyshev2") Then
      Print #1, "[b,a]=cheby2(" + Order + "," + ripple2 + ",[" + lower + _
    "," + upper + "],'stop','s')"
    Print #1, "freqs(b,a)"
    Else
    Print #1, "[b,a]=ellip(" + Order + "," + ripple1 + "," + ripple2 + _
    ",[" + lower + "," + upper + "],'stop', 's')"
    Print #1, "freqs(b,a)"
    End If
  End If
End If
End Select
Close ' close the file
 Dim stAppName As String
 ' lunch Matlab with the Figure display the filter response
    stAppName = "C:\Matlab-complete\bin\win32\matlab.exe /r case2"
    Call Shell(stAppName, 2)
End Sub
```

```
Private Sub cmdSchema_Click( )
' this to lunch Schematica filter design tool
On Error GoTo Err_Command1_Click
   Dim stAppName As String
   stAppName = "C:\Program Files\Filter Wiz PRO 3.2\FWP3232.exe"
   Call Shell(stAppName, 1)
Exit_Command1_Click:
   Exit Sub
Err_Command1_Click:
   MsgBox Err.Description
   Resume Exit_Command1_Click
End Sub
```

```
Private Sub cmdFilterCad_Click( )
' to lunch FilterCad filter design tool
On Error GoTo Err_Command2_Click
   Dim stAppName As String
   stAppName = "C:\Program Files\LTC\FilterCAD\FilterCAD.exe"
   Call Shell(stAppName, 1)
Exit_Command2_Click:
   Exit Sub
Err_Command2_Click:
   MsgBox Err.Description
   Resume Exit_Command2_Click

End Sub
```

```
Private Sub cmdNew_Click( )
' returns to a form where new custom design can take place
Unload Form1
Unload Form2
frmTypeTech.Show
End Sub
```

```
Private Sub cmdModify_Click( )
' click when user intend to modify a design
Frame2.Visible = True
cmdModify.Visible = False
End Sub


Private Sub DatApp_Reposition( )
'this event to calculate the % match based on data in the current record
DatApp.Caption = DatApp.Recordset.AbsolutePosition + 1 & " of " & _
DatApp.Recordset.RecordCount
Dim inputVolume ' declare the optimising criteria
Dim inputCost
Dim inputComplexity
Dim inputCount
inputVolume = CboVolume.Text
inputCost = CboCost.Text
inputCount = CboCount.Text
inputComplexity = CboComplexity.Text
' calculate the total score for the match
' by comparing the selected criteria to the existing one in the database
Select Case inputVolume ' start by first crieria or volume
 Case "low"
  If txtVolume.Text = "low" Then
   txtScore.Text = 10
  Else
   If txtVolume.Text = "medium" Then
    txtScore.Text = "5"
   Else
    txtScore.Text = 2
   End If
  End If
 Case "medium"
  If txtVolume.Text = "low" Then
   txtScore.Text = 5
  Else
   If txtVolume.Text = "medium" Then
    txtScore.Text = 10
```

```
  Else
   txtScore.Text = 5
  End If
 End If
Case "high"
 If txtVolume.Text = "low" Then
  txtScore.Text = 2
 Else
  If txtVolume.Text = "medium" Then
   txtScore.Text = 5
  Else
   txtScore.Text = 10
  End If
 End If
End Select
Select Case inputCost ' design criteria of cost
 Case "low"
  If txtCost.Text = "low" Then
   txtScoreC.Text = 10
  Else
   If txtCost.Text = "medium" Then
    txtScoreC.Text = 6
   Else
    txtScoreC.Text = 2
   End If
  End If
 Case "medium"
  If txtCost.Text = "low" Then
   txtScoreC.Text = 4
  Else
   If txtCost.Text = "medium" Then
    txtScoreC.Text = 10
   Else
    txtScoreC.Text = 6
   End If
  End If
 Case "high"
  If txtCost.Text = "low" Then
   txtScoreC.Text = 3
  Else
```

```
  If txtCost.Text = "medium" Then
   txtScoreC.Text = 6
  Else
   txtScoreC.Text = 10
  End If
 End If
End Select
Select Case inputComplexity ' design criteria of complexity
 Case "low"
  If txtComplexity.Text = "low" Then
   txtScoreCom.Text = 10
  Else
   If txtComplexity.Text = "medium" Then
    txtScoreCom.Text = 6
   Else
    txtScoreCom.Text = 2
   End If
  End If
 Case "medium"
  If txtComplexity.Text = "low" Then
   txtScoreCom.Text = 6
  Else
   If txtComplexity.Text = "medium" Then
    txtScoreCom.Text = 10
   Else
    txtScoreCom.Text = 4
   End If
  End If
 Case "high"
  If txtComplexity.Text = "low" Then
   txtScoreCom.Text = 2
  Else
   If txtComplexity.Text = "medium" Then
    txtScoreCom.Text = 6
   Else
    txtScoreCom.Text = 10
   End If
  End If
End Select
Select Case inputCount ' design criteria of component count
```

```
Case "<10"
 If Val(txtCount.Text) < 10 Then
  txtScoreCount.Text = 10
 Else
  If Val(txtCount.Text) >= 10 And Val(txtCount.Text) <= 20 Then
   txtScoreCount.Text = 7
  Else
   If Val(txtCount.Text) >= 21 And Val(txtCount.Text) <= 30 Then
   txtScoreCount.Text = 4
   Else
     txtScoreCount.Text = 1
   End If
  End If
 End If
  Case "10-20"
 If Val(txtCount.Text) < 10 Then
  txtScoreCount.Text = 8
 Else
  If Val(txtCount.Text) >= 10 And Val(txtCount.Text) <= 20 Then
   txtScoreCount.Text = 10
  Else
   If Val(txtCount.Text) >= 21 And Val(txtCount.Text) <= 30 Then
    txtScoreCount.Text = 8
   Else
   txtScoreCount.Text = 3
   End If
  End If
 End If
Case "21-30"
 If Val(txtCount.Text) < 10 Then
  txtScoreCount.Text = 4
 Else
  If Val(txtCount.Text) >= 10 And Val(txtCount.Text) <= 20 Then
   txtScoreCount.Text = 8
   Else
   If Val(txtCount.Text) >= 21 And Val(txtCount.Text) <= 30 Then
    txtScoreCount.Text = 10
   Else
     txtScoreCount.Text = 6
   End If
```

```
 End If
End If
Case ">30"
 If Val(txtCount.Text) < 10 Then
  txtScoreCount.Text = 2
  Else
  If Val(txtCount.Text) >= 10 And Val(txtCount.Text) <= 20 Then
   txtScoreCount.Text = 6
   Else
    If Val(txtCount.Text) >= 21 And Val(txtCount.Text) <= 30 Then
     txtScoreCount.Text = 8
    Else
     txtScoreCount.Text = 10
    End If
   End If
  End If
End Select
' calculate the total % score of the current record
txtTotalScore.Text = Val(txtScore.Text) + Val(txtScoreC.Text) _
+ Val(txtScoreCom.Text) + Val(txtScoreCount.Text)
txtTotalScore.Text = Val(txtTotalScore.Text) * 2.5
End Sub


Private Sub Form_Activate( )
' to make first record as a current
DatApp.Recordset.MoveLast
DatApp.Recordset.MoveFirst
End Sub
```

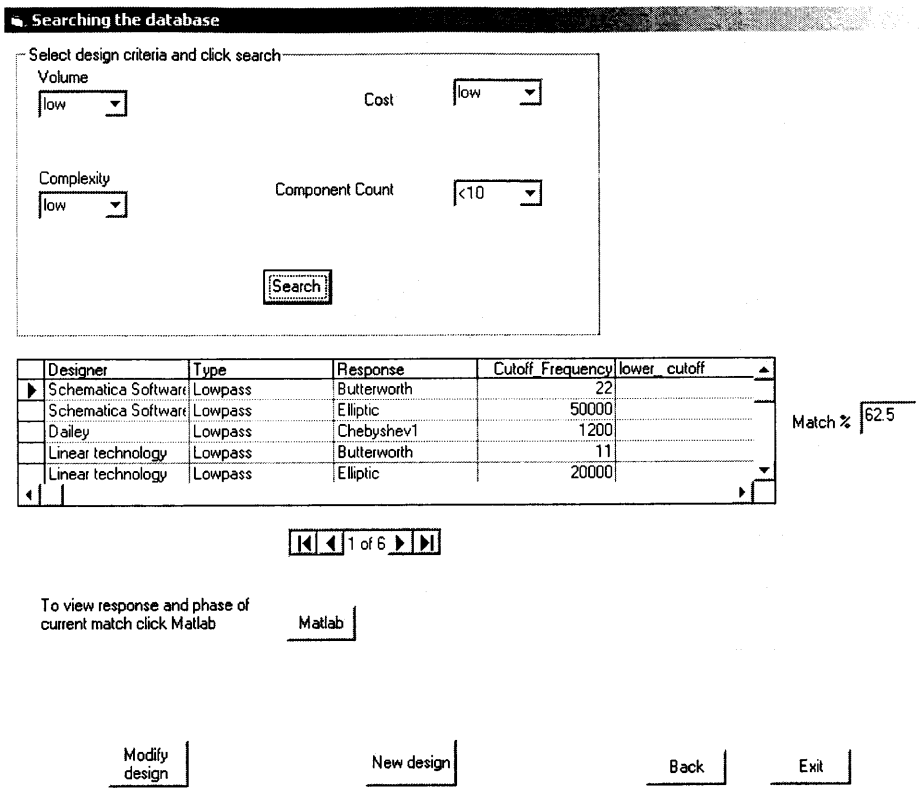


Figure A4.7: *Form of "Searching the database"*

The code for the form in Fig. A4.7:

Private Sub cmdBack_Click( )

'back to new design form

Unload Me

End Sub

Private Sub cmdMatlab_Click( )

'Lunch Matlab to display the frequency and phase response

'for the current record

Dim Order ' declare the filter variables

Dim Cutoff

Dim response

Dim ripple1

Dim ripple2

Dim filtertype

Dim upper

```
Dim lower
Order = txtOrder.Text
Cutoff = txtCutoff.Text
response = txtResponse.Text
ripple1 = txtPassRipple.Text
ripple2 = txtStopRipple.Text
filtertype = txtType.Text
upper = txtUpper.Text
lower = txtLower.Text
Open "C:\Matlab-complete\work\case2.m" For Output As 1
'open file to store filter variables
Select Case filtertype
'write filter variables into the file according to each filter type
Case "Lowpass"
If (response = "Butterworth") Then

 Print #1, "[b,a]=butter(" + Order + "," + Cutoff + ",'s')"
 Print #1, "freqs(b,a)"
 Else
   If (response = "Chebyshev1") Then
   Print #1, "[b,a]=cheby1(" + Order + "," + ripple1 + "," + _
 Cutoff + ", 's')"
 Print #1, "freqs(b,a)"
   Else
 If (response = "Chebyshev2") Then
    Print #1, "[b,a]=cheby2(" + Order + "," + ripple2 + "," + _
 Cutoff + ", 's')"
 Print #1, "freqs(b,a)"
 Else
 Print #1, "[b,a]=ellip(" + Order + "," + ripple1 + "," + ripple2 _
 + "," + Cutoff + ", 's')"
 Print #1, "freqs(b,a)"
 End If
 End If
End If
Case "Highpass"
If (response = "Butterworth") Then
 Print #1, "[b,a]=butter(" + Order + "," + Cutoff + ",'high', 's')"
 Print #1, "freqs(b,a)"
 Else
```

```
  If (response = "Chebyshev1") Then
  Print #1, "[b,a]=cheby1(" + Order + "," + ripple1 + "," + _
 Cutoff + ",'high', 's')"
 Print #1, "freqs(b,a)"
  Else
 If (response = "Chebyshev2") Then
  Print #1, "[b,a]=cheby2(" + Order + "," + ripple2 + "," + _
 Cutoff + ",'high', 's')"
 Print #1, "freqs(b,a)"
 Else
 Print #1, "[b,a]=ellip(" + Order + "," + ripple1 + "," + ripple2 _
 + "," + Cutoff + ",'high', 's')"
 Print #1, "freqs(b,a)"
  End If
 End If
End If
Case "Bandpass"
If (response = "Butterworth") Then
 Print #1, "[b,a]=butter(" + Order + ",[" + lower + "," + upper _
+ "], 's')"
Print #1, "freqs(b,a)"
Else
  If (response = "Chebyshev1") Then
  Print #1, "[b,a]=cheby1(" + Order + "," + ripple1 + ",[" + lower _
+ "," + upper + "], 's')"
Print #1, "freqs(b,a)"
  Else
 If (response = "Chebyshev2") Then
  Print #1, "[b,a]=cheby2(" + Order + "," + ripple2 + ",[" + lower _
 + "," + upper + "], 's')"
 Print #1, "freqs(b,a)"
 Else
 Print #1, "[b,a]=ellip(" + Order + "," + ripple1 + "," + ripple2 _
 + ",[" + lower + "," + upper + "], 's')"
 Print #1, "freqs(b,a)"
 End If
End If
End If
Case Else
If (response = "Butterworth") Then
```

```
 Print #1, "[b,a]=butter(" + Order + ",[" + lower + "," + upper _
+ "],'stop', 's')"
Print #1, "freqs(b,a)"
Else
  If (response = "Chebyshev1") Then
  Print #1, "[b,a]=cheby1(" + Order + "," + ripple1 + ",[" + lower _
+ "," + upper + "],'stop', 's')"
Print #1, "freqs(b,a)"
  Else
 If (response = "Chebyshev2") Then
   Print #1, "[b,a]=cheby2(" + Order + "," + ripple2 + ",[" + lower _
 + "," + upper + "],'stop','s')"
 Print #1, "freqs(b,a)"
 Else
 Print #1, "[b,a]=ellip(" + Order + "," + ripple1 + "," + ripple2 _
 + ",[" + lower + "," + upper + "],'stop', 's')"
 Print #1, "freqs(b,a)"
  End If
End If
End If
End Select
Close   ' close the file
 Dim stAppName As String
 ' lunch Matlab with the Figure display the filter response
   stAppName = "C:\Matlab-complete\bin\win32\matlab.exe /r case2"
   Call Shell(stAppName, 2)
End Sub
```

```
Private Sub CmdSearch_Click( )
'display the retrieved record on the FlexGrid
' also to display the percentage match for the current record

DatApp.RecordSource = "SELECT * from filter_cases where Type = '" & _
            frmTypeTech.CboType.Text & "'"
DatApp.Refresh
cmdexit.Visible = True
cmdModify.Visible = True
cmdSchema.Visible = True
cmdFilterCad.Visible = True
```

```
cmdNew.Visible = True
cmdBack.Visible = True
DBGrid1.Visible = True
DatApp.Visible = True
cmdMatlab.Visible = True
Label1.Visible = True
Label4.Visible = True
txtTotalScore.Visible = True
End Sub


Private Sub cmdNew_Click( )
' returns to a form where new custom design can take place
frmNew.Show
End Sub


Private Sub DatApp_Reposition( )
'this event to calculate the % match based on data in the current record
DatApp.Caption = DatApp.Recordset.AbsolutePosition + 1 & " of " & _
DatApp.Recordset.RecordCount
Dim inputVolume ' declare the optimising criteria
Dim inputCost
Dim inputComplexity
Dim inputCount
inputVolume = CboVolume.Text
inputCost = CboCost.Text
inputCount = CboCount.Text
inputComplexity = CboComplexity.Text
' calculate the total score for the match
' by comparing the selected criteria to the existing one in the database
Select Case inputVolume ' start by first crieria or volume
 Case "low"
  If txtVolume.Text = "low" Then
   txtScore.Text = 10
  Else
   If txtVolume.Text = "medium" Then
    txtScore.Text = "5"
   Else
```

```
  txtScore.Text = 2
  End If
 End If
Case "medium"
 If txtVolume.Text = "low" Then
  txtScore.Text = 5
 Else
  If txtVolume.Text = "medium" Then
   txtScore.Text = 10
  Else
   txtScore.Text = 5
  End If
 End If
Case "high"
 If txtVolume.Text = "low" Then
  txtScore.Text = 2
 Else
  If txtVolume.Text = "medium" Then
   txtScore.Text = 5
  Else
   txtScore.Text = 10
  End If
 End If
End Select
Select Case inputCost   ' design criteria of cost
 Case "low"
 If txtCost.Text = "low" Then
  txtScoreC.Text = 10
 Else
  If txtCost.Text = "medium" Then
   txtScoreC.Text = 6
  Else
   txtScoreC.Text = 2
  End If
 End If
Case "medium"
 If txtCost.Text = "low" Then
  txtScoreC.Text = 4
 Else
  If txtCost.Text = "medium" Then
```

```
    txtScoreC.Text = 10
   Else
    txtScoreC.Text = 6
   End If
  End If
 Case "high"
  If txtCost.Text = "low" Then
   txtScoreC.Text = 3
  Else
   If txtCost.Text = "medium" Then
    txtScoreC.Text = 6
   Else
    txtScoreC.Text = 10
   End If
  End If
End Select
Select Case inputComplexity ' design criteria of complexity
 Case "low"
  If txtComplexity.Text = "low" Then
   txtScoreCom.Text = 10
  Else
   If txtComplexity.Text = "medium" Then
    txtScoreCom.Text = 6
   Else
    txtScoreCom.Text = 2
   End If
  End If
 Case "medium"
  If txtComplexity.Text = "low" Then
   txtScoreCom.Text = 6
  Else
   If txtComplexity.Text = "medium" Then
    txtScoreCom.Text = 10
   Else
    txtScoreCom.Text = 4
   End If
  End If
 Case "high"
  If txtComplexity.Text = "low" Then
   txtScoreCom.Text = 2
```

```
 Else
  If txtComplexity.Text = "medium" Then
   txtScoreCom.Text = 6
  Else
   txtScoreCom.Text = 10
  End If
 End If
End Select
Select Case inputCount  ' design criteria of component count
 Case "<10"
  If Val(txtCount.Text) < 10 Then
   txtScoreCount.Text = 10
  Else
   If Val(txtCount.Text) >= 10 And Val(txtCount.Text) <= 20 Then
    txtScoreCount.Text = 7
   Else
    If Val(txtCount.Text) >= 21 And Val(txtCount.Text) <= 30 Then
    txtScoreCount.Text = 4
    Else
      txtScoreCount.Text = 1
    End If
   End If
  End If
  Case "10-20"
  If Val(txtCount.Text) < 10 Then
   txtScoreCount.Text = 8
  Else
   If Val(txtCount.Text) >= 10 And Val(txtCount.Text) <= 20 Then
    txtScoreCount.Text = 10
   Else
    If Val(txtCount.Text) >= 21 And Val(txtCount.Text) <= 30 Then
     txtScoreCount.Text = 8
    Else
    txtScoreCount.Text = 3
    End If
   End If
  End If
 Case "21-30"
  If Val(txtCount.Text) < 10 Then
   txtScoreCount.Text = 4
```

```
 Else
  If Val(txtCount.Text) >= 10 And Val(txtCount.Text) <= 20 Then
   txtScoreCount.Text = 8
   Else
   If Val(txtCount.Text) >= 21 And Val(txtCount.Text) <= 30 Then
    txtScoreCount.Text = 10
   Else
      txtScoreCount.Text = 6
   End If
  End If
 End If
 Case ">30"
  If Val(txtCount.Text) < 10 Then
   txtScoreCount.Text = 2
   Else
   If Val(txtCount.Text) >= 10 And Val(txtCount.Text) <= 20 Then
    txtScoreCount.Text = 6
    Else
     If Val(txtCount.Text) >= 21 And Val(txtCount.Text) <= 30 Then
      txtScoreCount.Text = 8
     Else
      txtScoreCount.Text = 10
     End If
    End If
   End If
End Select
' calculate the total % score of the current record
txtTotalScore.Text = Val(txtScore.Text) + Val(txtScoreC.Text) + _
Val(txtScoreCom.Text) + Val(txtScoreCount.Text)
txtTotalScore.Text = Val(txtTotalScore.Text) * 2.5
End Sub


Private Sub cmdExit_Click( )
Const title = "Closing the application"
Dim prompt  As String
Dim msgType As Integer
' confirm the exit request
prompt = "This will close the application."
prompt = prompt & Chr(13) & Chr(13) & "Are you sure ?"
```

```
msgType = vbCritical + vbYesNo + vbDefaultButton2
If MsgBox(prompt, msgType, title) = vbNo Then
   Cancel = True
Else
Unload Me
Unload frmdesign
Unload frmTypeTech
Unload Form1
Unload Form2
End If
End Sub


Private Sub cmdModify_Click( )
' click when user intend to modify a design
Frame3.Visible = True
cmdModify.Visible = False
End Sub


Private Sub cmdSchema_Click( )
' this to lunch Schematica filter design tool
On Error GoTo Err_cmdSchema_Click
   Dim stAppName As String
   stAppName = "C:\Program Files\Filter Wiz PRO 3.2\FWP3232.exe"
   Call Shell(stAppName, 1)
Exit_cmdSchema_Click:
   Exit Sub
Err_cmdSchema_Click:
   MsgBox Err.Description
   Resume Exit_cmdSchema_Click
End Sub


Private Sub cmdFilterCad_Click( )
' to lunch FilterCad filter design tool
On Error GoTo Err_cmdFilterCad_Click
   Dim stAppName As String
   stAppName = "C:\Program Files\LTC\FilterCAD\FilterCAD.exe"
```

```
    Call Shell(stAppName, 1)
Exit_cmdFilterCad_Click:
    Exit Sub
Err_cmdFilterCad_Click:
    MsgBox Err.Description
    Resume Exit_cmdFilterCad_Click
End Sub


Private Sub Form_Activate( )
' to make first record as a current
DatApp.Recordset.MoveLast
DatApp.Recordset.MoveFirst
End Sub


Private Sub Form_Load( )
' show the frame of setting criteria
Frame1.Visible = True
End Sub
```

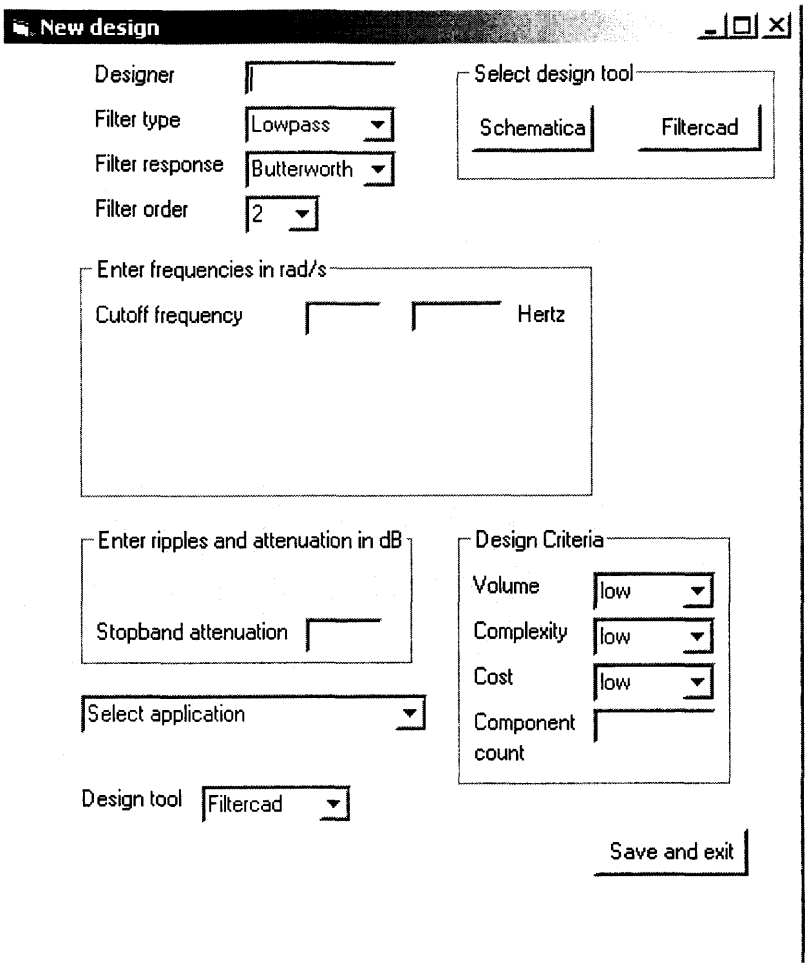


Figure A4.8: *Form of "New design"*

The code for the form in Fig. A4.8:

```
Private Sub cboResponse_click( )
'set the response combo box
txtPassRipple.Visible = True
txtStopRipple.Visible = True
Label5.Visible = True
Label6.Visible = True
fraRipples.Visible = True
If cboResponse.Text = "Butterworth" Then
   txtPassRipple.Visible = False
   Label5.Visible = False
End If
```

```
If cboResponse.Text = "Chebyshev2" Then
   txtPassRipple.Visible = False
   Label5.Visible = False
End If
End Sub


Private Sub CboType_click( )
' set the filter type and the associated fields
txtCutoff.Visible = True
txtLower.Visible = True
txtUpper.Visible = True
Label2.Visible = True
Label3.Visible = True
Label4.Visible = True
lblCutoff.Visible = True
lblLower.Visible = True
lblUpper.Visible = True
Label9.Visible = True
Label11.Visible = True
Label13.Visible = True
If CboType.Text = "Lowpass" Or CboType.Text = "Highpass" Then
   txtLower.Visible = False
   txtUpper.Visible = False
   Label3.Visible = False
   Label4.Visible = False
   lblLower.Visible = False
   lblUpper.Visible = False
   Label11.Visible = False
   Label13.Visible = False
Else
   If CboType.Text = "Bandpass" Or CboType.Text = "Bandstop" Then
      txtCutoff.Visible = False
      Label2.Visible = False
      lblCutoff.Visible = False
      Label9.Visible = False
   End If
End If
End Sub
```

```
Private Sub cmdGo_Click( )
'to add the new design record into the recordset
If DatNew.Recordset.EditMode = dbEditNone Then Exit Sub
'check the field data and its vaildation
If txtDesigner.Text = "" Then
'check the designer field
   MsgBox " Please enter designer name"
   txtDesigner.SetFocus
   Exit Sub
End If
If CboType.Text = "" Then
' check the filter type field
   MsgBox "Please select filter type"
   CboType.SetFocus
   Exit Sub
End If
If cboResponse.Text = "" Then
'check the filter response field
   MsgBox "Please select filter response"
   cboResponse.SetFocus
   Exit Sub
End If
If cboOrder.Text = "" Then
'check the filter order field
   MsgBox " Please select filter order"
   cboOrder.SetFocus
   Exit Sub
End If
' check the filter frequency fields
If txtCutoff.Visible = True And txtCutoff.Text = "" Then
   MsgBox "Please enter the Cutoff frequency"
   txtCutoff.SetFocus
   Exit Sub
End If
If txtLower.Visible = True And txtLower.Text = "" Then
   MsgBox "Please enter the lower cutoff frequency"
   txtLower.SetFocus
   Exit Sub
End If
If txtUpper.Visible = True And txtUpper.Text = "" Then
```

```
    MsgBox "Please enter the upper cutoff frequency"
    txtUpper.SetFocus
    Exit Sub
End If
If Val(txtLower.Text) > Val(txtUpper.Text) Then
    MsgBox "upper cutoff frequency must be greater than lower one"
    txtUpper.SetFocus
    Exit Sub
End If
'check the ripple fields
If txtPassRipple.Visible = True And txtPassRipple.Text = "" Then
    MsgBox " Please enter the passband ripple"
    txtPassRipple.SetFocus
    Exit Sub
End If
If txtStopRipple.Visible = True And txtStopRipple.Text = "" Then
    MsgBox " Please enter the stopband attenuation"
    txtStopRipple.SetFocus
    Exit Sub
End If
'check the application field
If CboApp.Text = "" Then
    MsgBox "Please select filter application"
    CboApp.SetFocus
    Exit Sub
End If
' check the design criteria fields
If CboVolume.Text = "" Then
    MsgBox " Please select volume"
    CboVolume.SetFocus
    Exit Sub
End If
If CboComplexity.Text = "" Then
    MsgBox " Please select design complexity"
    CboComplexity.SetFocus
    Exit Sub
End If
If CboCost.Text = "" Then
    MsgBox "Please select cost"
    CboCost.SetFocus
```

```
    Exit Sub
End If
If txtCount.Text = "" Then
    MsgBox " Please enter component count"
    txtCount.SetFocus
    Exit Sub
End If
'check the selection of design tool
If cbotool.Text = "" Then
    MsgBox "Please select design tool"
    cbotool.SetFocus
    Exit Sub
End If
'edit and update the record
If DatNew.Recordset.EditMode = dbEditAdd Then
    DatNew.Recordset.Update
Else
    DatNew.Recordset.Edit
    DatNew.Recordset.Update
End If
Unload Me ' exit the form
frmSplash.Show
End Sub
```

```
Private Sub cmdSchem_Click( )
' to launch Schematica filter design tool
On Error GoTo Err_Command1_Click
    Dim stAppName As String
    stAppName = "C:\Program Files\Filter Wiz PRO 3.2\FWP3232.exe"
    Call Shell(stAppName, 1)
Exit_Command1_Click:
    Exit Sub
Err_Command1_Click:
    MsgBox Err.Description
    Resume Exit_Command1_Click
End Sub
```

```
Private Sub cmdFilterCad_Click( )
' to launch FilterCad filter design tool
On Error GoTo Err_Command2_Click
   Dim stAppName As String
   stAppName = "C:\Program Files\LTC\FilterCAD\FilterCAD.exe"
   Call Shell(stAppName, 1)
Exit_Command2_Click:
   Exit Sub
Err_Command2_Click:
   MsgBox Err.Description
   Resume Exit_Command2_Click
End Sub
Private Sub Form_Activate()
'to make the form available when add new record
DatNew.Recordset.AddNew ' use the defaults
CboType.Text = "Lowpass"
cboResponse.Text = "Butterworth"
CboApp.Text = "Select application"
cboOrder.Text = "2"
cbotool.Text = "Filtercad"
CboCost.Text = "low"
CboVolume.Text = "low"
CboComplexity.Text = "low"
End Sub

Private Sub Form_Load( )
' to set controls and the associated controls on the form
txtCutoff.Visible = True
txtLower.Visible = True
txtUpper.Visible = True
Label2.Visible = True
Label3.Visible = True
Label4.Visible = True
lblCutoff.Visible = True
lblLower.Visible = True
lblUpper.Visible = True
Label9.Visible = True
Label11.Visible = True
Label13.Visible = True
```

```
If CboType.Text = "Lowpass" Or CboType.Text = "Highpass" Then
   txtLower.Visible = False
   txtUpper.Visible = False
   Label3.Visible = False
   Label4.Visible = False
   lblLower.Visible = False
   lblUpper.Visible = False
   Label11.Visible = False
   Label13.Visible = False
  Else
   If CboType.Text = "Bandpass" Or CboType.Text = "Bandstop" Then
      txtCutoff.Visible = False
      Label2.Visible = False
      lblCutoff.Visible = False
      Label9.Visible = False
   End If
End If
txtPassRipple.Visible = True
txtStopRipple.Visible = True
Label5.Visible = True
Label6.Visible = True
fraRipples.Visible = True
If cboResponse.Text = "Butterworth" Then
   txtPassRipple.Visible = False
   Label5.Visible = False
End If
If cboResponse.Text = "Chebyshev2" Then
   txtPassRipple.Visible = False
   Label5.Visible = False
End If
End Sub

Private Sub txtCutoff_Change( )
'check the cutoff frequency limit
Dim response
If Val(txtCutoff.Text) > 100000 Then
   response = MsgBox("Please enter frequency between 1 and 100000 rad/s", _
   vbOKOnly + vbExclamation, "Frequency limit")
End If
If response = vbOK Then
```

```
    txtCutoff.SetFocus
    txtCutoff.Text = " "
    lblCutoff.Caption = ""
End If
lblCutoff.Caption = (1 / 6.28) * Val(txtCutoff.Text)
lblCutoff.Visible = True
Label9.Visible = True
End Sub
```

```
Private Sub txtLower_Change( )
' check the lower frequency limit
Dim response1
If Val(txtLower.Text) > 100000 Then
    response1 = MsgBox("Please enter frequency between 1 and 100000 rad/s", _
    vbOKOnly + vbExclamation, "Frequency limit")
End If
If response1 = vbOK Then
    txtLower.SetFocus
    txtLower.Text = " "
    lblLower.Caption = " "
End If
    lblLower.Caption = (1 / 6.28) * Val(txtLower.Text)
    lblLower.Visible = True
    Label11.Visible = True
End Sub
```

```
Private Sub txtPassRipple_LostFocus( )
'check the range limit of passband ripple
Dim response3
If Val(txtPassRipple.Text) < 0.01 Or Val(txtPassRipple.Text) > 3 Then
   response3 = MsgBox("Please enter Passband ripple between 0.01 and 3 dB", _
   vbOKOnly + vbExclamation, "Passband Ripple")
     If response3 = vbOK Then
       txtPassRipple.SetFocus
       txtPassRipple.Text = " "
```

```
    End If
End If
End Sub
```

```
Private Sub txtStopRipple_LostFocus( )
'check the range limit of stopband ripple
Dim response4
If Val(txtStopRipple.Text) <= 3 Or Val(txtStopRipple.Text) > 100 Then
   response4 = MsgBox("Please enter Stopband ripple between 1 and 100 dB", _
   vbOKOnly + vbExclamation, "Passband Ripple")
      If response4 = vbOK Then
         txtStopRipple.SetFocus
         txtStopRipple.Text = " "
      End If
End If
End Sub
```

```
Private Sub txtUpper_change( )
'check the upper frequency limit
Dim response2
If Val(txtUpper.Text) > 100000 Then
   response2 = MsgBox("Please enter frequency between 1 and 100000 rad/s", vbOKOnly +
vbExclamation, "Frequency limit")
End If
If response2 = vbOK Then
   txtUpper.SetFocus
   txtUpper.Text = " "
   lblUpper.Caption = " "
End If
   lblUpper.Caption = (1 / 6.28) * Val(txtUpper.Text)
   lblUpper.Visible = True
   Label13.Visible = True
End Sub
```

Appendix B

Formulas for Filter Approximations[1,2]

The following equations are for lowpass filters normalised to a 3-dB cutoff of 1 rad/s.

B1. Butterworth Approximation

The attenuation of a Butterworth can be expressed by

$$A_{dB} = 10\log\left[1+\left(\frac{\omega_\chi}{\omega_c}\right)^{2n}\right]$$

Where ω_χ/ω_c is the ratio of the given frequency ω_χ to the 3-dB cutoff frequency ω_c and n is the order of the filter.

An example of 4th order, 1000rad/s Butterworth lowpass filter is shown in Fig. B1.1

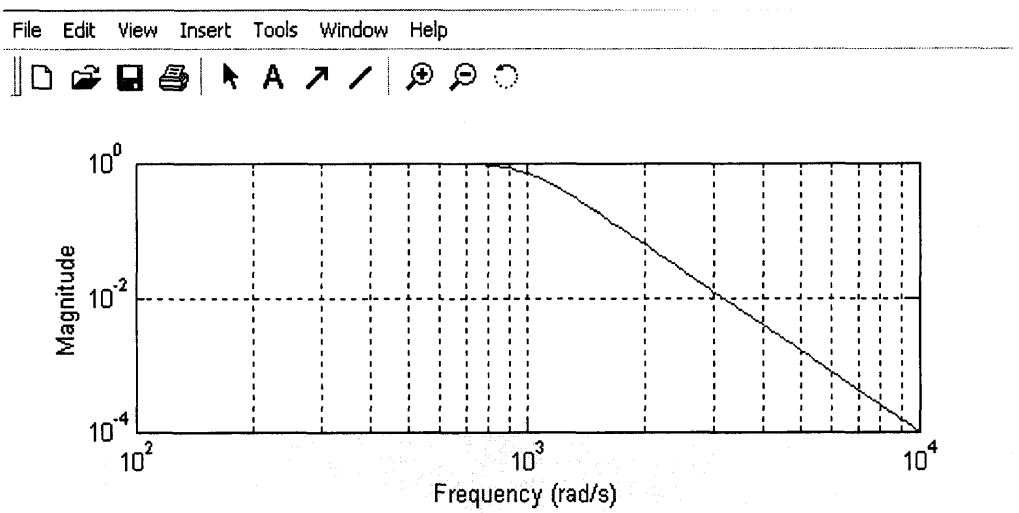


Fig B1.1: *4th order, 1000rad/s Butterworth lowpass filter*

B2. Chebyshev Approximation

The attenuation of Chebyshev filters can be expressed as

$$A_{dB} = 10\log\left[1+\varepsilon^2 C_n^2(\Omega)\right]$$

Where $C_n(\Omega)$ is a Chebyshev polynomial whose magnitude oscillates between ± 1 for $\Omega \leq 1$

And $$\varepsilon = \sqrt{10^{R_{dB}/10} - 1}$$

Where R_{dB} is the ripple in decibels.

An example of 4th order, 1000rad/s Chebyshev lowpass filter is shown in Fig. B2.1

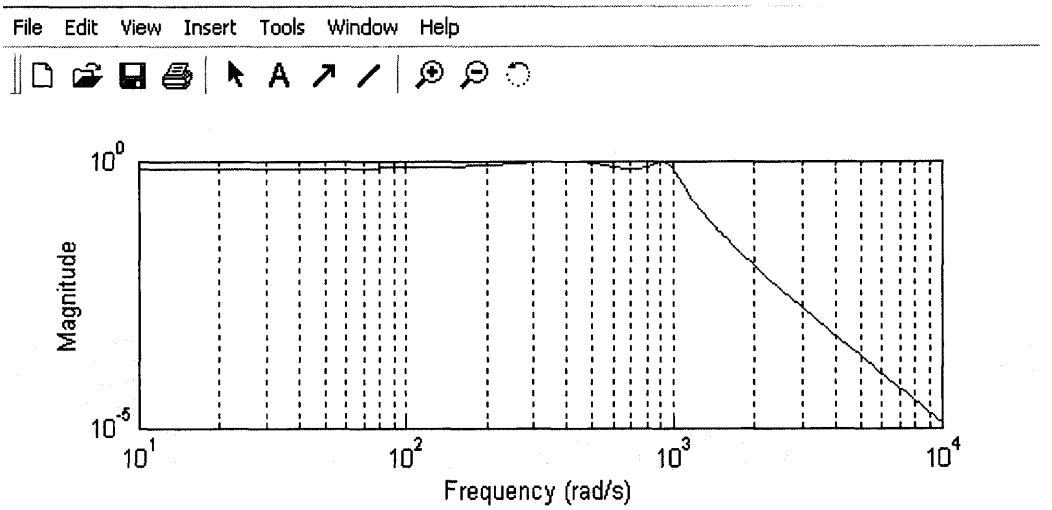


Fig B2.1: *4th order, 1000rad/s Chebyshev lowpass filter*

B3. Elliptic Approximation

The attenuation of elliptic filters can be expressed as

$$A_{dB} = 10\log\left[1+\varepsilon^2 Z_n^2(\Omega)\right]$$

Where ε is determined by the ripple and $Z_n(\Omega)$ is an elliptic function of the *n*th order.

Elliptic functions have both poles and zeros and can be expressed as

$$Z_n(\Omega) = \frac{\Omega\left(a_2{}^2-\Omega^2\right)\left(a_4{}^2-\Omega^2\right).....\left(a_m{}^2-\Omega^2\right)}{\left(1-a_2{}^2\Omega^2\right)\left(1-a_4{}^2\Omega^2\right).....\left(1-a_m{}^2\Omega^2\right)}$$

Where n is odd and $m = (n-1)/2$, or

$$Z_n(\Omega) = \frac{\left(a_2{}^2-\Omega\right)\left(a_4{}^2-\Omega^2\right).....\left(a_m{}^2-\Omega^2\right)}{\left(1-a_2{}^2\Omega^2\right)\left(1-a_4{}^2\Omega^2\right).....\left(1-a_m{}^2\Omega^2\right)}$$

Where n is even and $m = n/2$.

An example of 4th order, 1000rad/s Elliptic lowpass filter is shown in Fig. B3.1

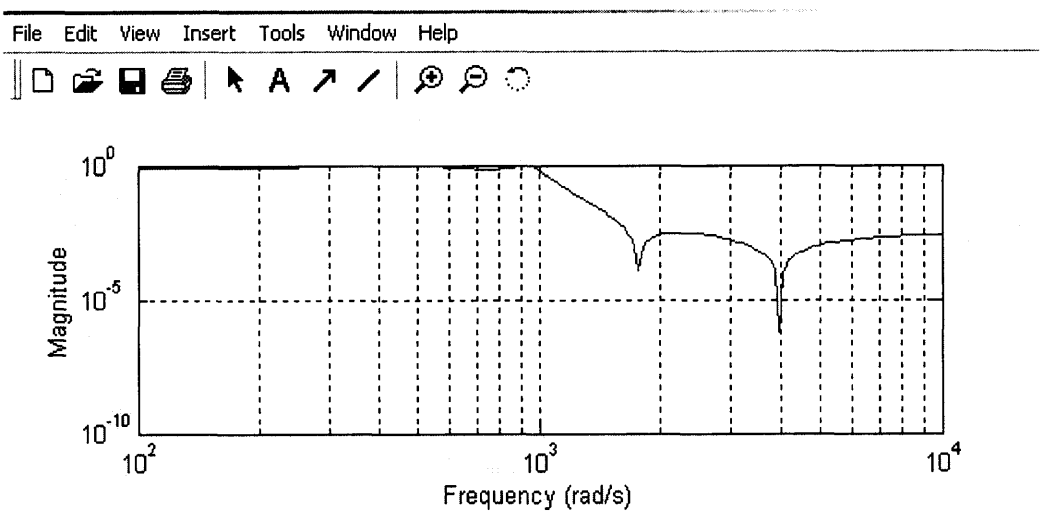


Fig B3.1: *4th order, 1000rad/s Elliptic lowpass filter*

[1] William, A. and Taylor, F. (1995). *Electronic filter design handbook.* 3rd edition, New York: McGraw-Hill.

[2] Matlab signal processing toolbox. (2002). The Mathworks, Inc.

Appendix C

Publications

1. S A El-Nakla and D A Bradley (2002). "An Expert System to Support Electronic Design Within Mechatronics", Proceedings of the 8th Mechatronics Forum International Conference, Enschede, the Netherlands.

2. S A El-Nakla and D A Bradley (2004). "Case-Based Reasoning and Conflict Resolution in support of the Design of Mechatronic Systems". 9th Mechatronics Forum International Conference. Ankara, Turkey.